Л. В. Кнауб, Е. А. Новиков, Ю. А. Шитов

ТЕОРЕТИКО-ЧИСЛЕННЫЕ МЕТОДЫ В КРИПТОГРАФИИ

Учебное пособие

институт фундаментальной подготовки

СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
SIBERIAN FEDERAL UNIVERSITY

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Л. В. Кнауб, Е. А. Новиков, Ю. А. Шитов

ТЕОРЕТИКО-ЧИСЛЕННЫЕ МЕТОДЫ В КРИПТОГРАФИИ

Рекомендовано Сибирским региональным учебно-методическим центром высшего профессионального образования для межвузовского использования в качестве учебного пособия для студентов, обучающихся по специальности 090102 «Компьютерная безопасность» и направлениям подготовки 090900 «Информационная безопасность» и 010200 «Математика и компьютерные науки» от 5 июля 2010 г.

Красноярск
СФУ
2011

УДК 519.72(075)
ББК 32.811я73
К53

Рецензенты:

И. О. Богульский, ведущий научный сотрудник ИВМ СОРАН, д-р физ.-мат. наук, проф.;

Ю. В. Шорников, д-р техн. наук, проф. кафедры АСУ новосибирского государственного технического университета

Кнауб, Л. В.

К53 Теоретико-численные методы в криптографии : учеб. пособие / Л. В. Кнауб, Е. А. Новиков, Ю. А. Шитов. – Красноярск : Сибирский федеральный университет, 2011. – 160 с.

ISBN 978-5-7638-2113-7

Излагаются некоторые элементы теории чисел, отношения сравнимости, модулярная арифметика, степенные вычеты, первообразные корни, индексы, алгоритмы дискретного логарифмирования, китайская теорема об остатках, простые числа и проверка на простоту, разложение чисел на множители и арифметические операции над большими числами. В прил. 1 описаны основы теории групп, колец и полей, а в прил. 2 приведены реализации некоторых алгоритмов, даны тексты программ на языке Borland C++, снабженные подробными комментариями.

Для студентов, обучающихся по специальности 090102 «Компьютерная безопасность» и направлениям подготовки 090900 «Информационная безопасность» и 010200 «Математика и компьютерные науки».

УДК 519.72(075)
ББК 32.811я73

Учебное издание

Кнауб Людмила Владимировна, Новиков Евгений Александрович
Шитов Юрий Александрович

ТЕОРЕТИКО-ЧИСЛЕННЫЕ МЕТОДЫ В КРИПТОГРАФИИ

Редактор Т. М. Пыжик
Компьютерная вёрстка Д. Р. Мифтахутдинова

Подписано в печать 06.06.2011. Печать плоская. Формат 60×84/16
Бумага офсетная. Усл. печ. л. 9,3. Тираж 100 экз. Заказ № 2481

Редакционно-издательский отдел Библиотечно-издательского комплекса
Сибирского федерального университета. 660041, г. Красноярск, пр. Свободный, 79

Отпечатано полиграфическим центром Библиотечно-издательского комплекса
Сибирского федерального университета. 660041, г. Красноярск, пр. Свободный, 82а

ISBN 978-5-7638-2113-7

ВВЕДЕНИЕ

Основой данного пособия является курс лекций по теоретико-численным методам в криптографии (ТЧМК). Этот курс предназначен для студентов ИКИТ кафедры прикладной математики и компьютерной безопасности, которые специализируются по информационной безопасности. Лекции по данному предмету формировались с 1996 года. Надо сказать, что в то время литературы на русском языке по таким направлениям, как защита информации, криптографии и математическим методам в криптографии практически не было (в отличии от зарубежных изданий). Поэтому на первом этапе «скелет» курса формировался на элементах теории чисел, алгоритмах арифметических операций с длинными числами и криптографических алгоритмах с открытыми ключами (RSA, схема Диффи – Хеллмана, схема Эль-Гамаля, схема аутентификации Шнора). Постепенно в тексты лекций, начиная с 2000 года, включались численные алгоритмы для решения трудных задач теории чисел. Для этого использовались переводы из зарубежных изданий по соответствующей тематике. В 2003 году Ю. А. Шитов издал методические указания по изучению численных алгоритмов для некоторых задач из теории чисел, которые использовались в курсе лекций по ТЧМК [15].

С 2001 года по направлениям «Криптография и математические методы в криптографии» начинают появляться учебные пособия и монографии на русском языке. В библиографическом списке приведен, по мнению авторов, достаточно полный обзор существующей литературы по этому направлению. В список не включены те учебники, в которых содержатся главы и разделы по арифметическим алгоритмам в теории чисел, так как любое учебное пособие по классическим курсам представляет собой аранжировку давно сформулированных и доказанных результатов, поэтому мы широко используем материалы из учебников, перечисленных в библиографическом списке.

Данное пособие отличается порядком и простотой изложения. Немного больше, чем в других пособиях, уделяется внимание решениям сравнений. Кроме того, при изложении результатов исключаются из рассмотрения доказательства некоторых трудных теорем, поскольку при желании слушатели курса могут ознакомиться с доказательствами в перечисленных источниках. При выборе материала авторы исходили из минимальных требований к начальной подготовке слушателей данного курса. Авторы предлагаемого пособия делают упор на возможность практической реализации изложенных алгоритмов на компьютере. Для некоторых алгоритмов, сформулированных в пособии, в приложении даны тексты программ, реализованных на языке BORLAND C++. Программы реализовал и тестировал М. В. Рыбков – студент группы ВТ 05–04, ИКИТ, СФУ.

Нумерация определений, теорем и соотношений приведены в разделах пособия.

НЕКОТОРЫЕ ЭЛЕМЕНТЫ ТЕОРИИ ЧИСЕЛ

В пособии не излагается теория чисел, а дан минимальный инструментарий из этой теории, который в дальнейшем потребуется для изучения криптографических систем, используемых в задачах по защите информации. Более подробное изложение математической теории чисел можно найти, например, в [1–2]. В теории чисел (по крайней мере, в той части, которая используется в криптографии) рассматривают целые числа и арифметические операции над ними. Перейдем к описанию основных понятий и определений из теории чисел [3].

Определение. Множество N натуральных чисел определяется с использованием аксиом Пеано:

1. $1 \in N$ (единица – натуральное число).

2. Для любого $a \in N$ существует единственное последующее $a^+ \in N$.

3. Для любого $a \in N$ выполняется неравенство $a^+ \neq 1$ (единица – наименьшее натуральное число).

4. Если $a^+ = b^+$, то $a = b$ (каждое последующее число обладает единственным предыдущим).

5. Если некоторое подмножество $N \subseteq \mathrm{N}$ содержит единицу и для каждого натурального числа $a \in N$ выполняется $a^+ \in N$, то $N = N$ (принцип индукции).

Таким образом, $N = \{1, 2, 3, 4, ...\}$.

На основании этих аксиом строится арифметика натуральных чисел, в которую включены операции сложения и умножения. Каждой паре натуральных чисел a, b можно единственным образом сопоставить их сумму – натуральное число $a + b$ так, чтобы выполнялись условия для любых натуральных чисел a, b, c:

1) $a + 1 = a^+$;

2) ассоциативность сложения: $(a + b) + c = a + (b + c)$;

3) коммутативность сложения: $a + b = b + a$;

4) если $a + b = a + c$, то $b = c$.

Каждой паре натуральных чисел a, b можно единственным образом сопоставить их произведение – натуральное число $a \cdot b$ так, чтобы выполнялись условия для любых натуральных чисел a, b, c:

1) $a \cdot 1 = a$;

2) $a \cdot b^+ = a \cdot b + a$;

3) ассоциативность умножения: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$;

4) коммутативность умножения: $a \cdot b = b \cdot a$;

5) дистрибутивность умножения относительно сложения:
$a \cdot (b + c) = a \cdot b + a \cdot c$, $(a + b) \cdot c = a \cdot c + b \cdot c$;

6) если $a \cdot b = a \cdot c$, то $b = c$.

Заметим, что множество натуральных чисел линейно упорядочено, т. е. для любых $a, b \in N$ выполняется ровно одно из условий: $a > b$, $a < b$, $a = b$.

Отношение < и > транзитивно, т. е. из неравенств $a < b$ и $b < c$ следует, что $a < c$. Если для $a, b \in N$ выполняется одно из соотношений $a > b$ или $a = b$, то записывают $a \le b$ или $b \ge a$.

Определение. Множество Z целых чисел определим как объединение множеств натуральных чисел, отрицательных натуральных чисел и нуля. Таким образом

$$Z = \{..., -3, -2, -1, 0, 1, 2, 3, ...\}.$$

На множестве Z целых чисел операции сложения и умножения задаются теми же правилами, что и для натуральных чисел.

Определение. Говорят, что целое число a делится (нацело) на целое число $b > 0$ (или что целое число $b > 0$ делит целое число a), если существует такое целое число c, что $a = bc$. Число a называют кратным числа b, число b – делителем числа a, число c – частным от деления a на b.

Пример. $8 = 4 \cdot 2$ (8 делится на 4 или 4 делит 8).

Отношение делимости обладает следующими свойствами.

1. Нуль делится на любое целое число.

2. Если a_1 делится на b, a_2 делится на b, то $a_1 + a_2$ делится на b.

3. Если $a_1 \pm a_2$ делится на b и a_1 делится на b, то a_2 делится на b.

4. Если a делится на b и x – произвольное целое число, то xa делится на b.

5. Любое целое число делится на 1.

6. Если a делится на b и b делится на c, то a делится на c.

7. Если 1 делится на a, то $a = \pm 1$.

Теорема 1. Для любого целого a и целого $b > 0$ существуют, и притом единственные, целые q и r, такие, что $a = bq + r$, $0 \le r < b$.

Число r называют остатком от деления a на b. Если $r = 0$, то число q называют полным частным, в противном случае ($r \ne 0$) число q называют неполным частным.

Доказательство. Возьмем числа $b \cdot 1$, $b \cdot 2$, …, … При $a \ge 0$ среди этих чисел рассмотрим множество M тех, которые больше, чем a. Согласно аксиом Пеано, множество M не пусто. Следовательно, в этом множестве существует некоторое наименьшее число $b \cdot k$. Тогда справедливо неравенство $b \cdot k \le a < b \cdot (k + 1)$.

При $a < 0$, $-a > 0$ можно рассмотреть множество M чисел $b \cdot i$ ($i = 1, 2, \ldots$), для которых выполняется неравенство $b \cdot i \ge -a$ и, соответственно, найти такое наименьшее число $b \cdot t'$, что при $t = t' - 1$ будет справедливо

$$bt < -a \le b(t + 1),$$

так что

$$b(-t-1) \le a < b(-t).$$

Итак, для a и b ($b > 0$) существует целое q, такое, что

$$bq \le a < b(q+1).$$

В этом случае, обозначив через r разность $a–bq$, получаем

$$r = a - bq < b(q+1) - bq = b,\ r \ge 0,$$

так что $a = bq + r$, $0 \le r < b$.

Определение. Целое число $d \neq 0$ называется общим делителем целых чисел $a_1, a_2,\ldots a_k$, если каждое из чисел $a_1, a_2,\ldots a_k$ делится на d.

Пример. Числа 2, 3, 6 являются общими делителями чисел 12, 18, 36, 72.

Определение. Общий делитель $d \neq 0$ чисел $a_1, a_2,\ldots a_k$ называется наибольшим общим делителем (НОД) этих чисел, если d делится на любой другой общий делитель этих чисел.

Наибольший общий делитель чисел $a_1, a_2,\ldots a_k$ обозначается так: d = НОД($a_1, a_2,\ldots a_k$) или просто $d = (a_1, a_2,\ldots a_k)$.

Пример. Число 6 является наибольшим делителями чисел 12, 18, 36, 72, т. е. 6 = НОД (12, 18, 36, 72) или 6 = (12, 18, 36, 72).

Теорема. Для любых целых чисел, из которых хотя бы одно отлично от нуля, существует единственный НОД (см., например, [3]).

Теорема 2. Для любых целых чисел $a_1, a_2, ..., a_k$ наибольший общий делитель d можно представить в виде линейной комбинации этих чисел, т. е.

$$d = c_1a_1 + c_2a_2 + ... + c_ka_k,$$

где c_i – целые числа.

Доказательство [3]. Пусть $A = \{c_1a_1 + c_2a_2 + ... + c_ka_k\}$ – множество всевозможных целочисленных линейных комбинаций чисел $a_1, a_2, ..., a_k$. Будем считать, что не все числа $a_1, a_2, ..., a_k$ нулевые, тогда в множестве A существует наименьший положительный элемент. Обозначим его через d. Покажем, что множество A совпадает с множеством всех целых чисел, кратных d. Поскольку число d может быть представлено в виде линейной комбинации чисел $a_1, a_2, ..., a_k$, то и любое число вида x_id, где x_i – целые числа, может быть представлено в виде линейной комбинации этих чисел.

Обратно, любая линейная комбинация чисел $a_1, a_2, ..., a_k$ делится на d. Действительно, применим к числу d и произвольному элементу $y \in A$ теорему о делении с остатком: существуют такие целые числа q и r, что $y = qd + r$, причем $0 < r < d$. Число $r = y – qd$ является элементом множества A,

поскольку $y \in A$ и $d \in A$. Но d – наименьший неотрицательный элемент множества A, значит, $r = 0$ и $y = qd$. Таким образом, $A = \{x_i d\}$.

Осталось доказать, что d = НОД(a_1, a_2,… a_k). Каждое из этих чисел имеет вид $x_i d$, (x_i – целое число), что следует из комбинации вида

$$0 \cdot a_1 + 0 \cdot a_2 + ... + 0 \cdot a_{i-1} + 1 \cdot a_i + 0 \cdot a_{i+1}, + ... + 0 \cdot a_k).$$

Таким образом, d – общий делитель чисел a_1, a_2, ..., a_k. Пусть c – любой другой делитель этих чисел. Тогда по свойству 2 делимости c делит каждую линейную комбинацию вида $c_1a_1 + c_2a_2 + ... + c_ka_k$, в том числе и ту, которая равна d.

Из доказанной теоремы следует, что для любых не равных одновременно 0 чисел a, b существует два числа c, d такие, что НОД(a, b) = $= a \cdot c + b \cdot d$.

Определение. Целое число n называют простым, если $|n| \neq 1$ и единственными делителями числа n являются числа 1, –1, n, $-n$.

Если натуральное число p не делится ни на одно простое число $d \leq \sqrt{p}$, то число p – простое.

Определение. Натуральное число $p > 1$ называется составным, если число p имеет, по крайней мере, один положительный делитель, отличный от 1 и p.

Если число p составное, то справедлива следующая теорема.

Теорема. Для любого составного числа наименьший, отличный от 1, положительный делитель является простым числом.

Определение. Целые числа a_1, a_2, ..., a_k называются взаимно простыми в совокупности, если НОД(a_1, a_2, ..., a_k) = 1. Целые числа a и b называются взаимно простыми, если НОД(a, b) = 1.

Определение. Целые числа a_1, a_2, ..., a_k называются попарно взаимно простыми, если НОД(a_1, a_j) = 1 для всех $1 \leq i \neq j \leq k$.

Пример. Числа 3, 15, 6, 4 взаимно просты в совокупности, так как НОД(3, 15, 6, 4) = 1. Числа 3, 8, 17, 25, 121 попарно взаимно просты.

Теорема. Для того чтобы целые числа были взаимно просты, необходимо и достаточно, чтобы существовали такие целые числа,что $a \cdot c + b \cdot d = 1$.

Теорема 3 (основная теорема арифметики [3]). Всякое целое число n ($n \neq -1, 0, 1$) можно представить в виде $n = \varepsilon\, p_1\, p_2 ... p_r$, где $\varepsilon = \pm 1$ и p_1 ,p_2 ,..., p_r – простые числа (не обязательно различные), $r \geq 1$. Это представление единственно с точностью до порядка сомножителей.

Доказательство [3]. Сначала докажем, что разложение существует. Пусть n – целое число ($n \neq -1, 0, 1$). Доказывать будем индукцией по $|n|$. Если $|n| = 2$, то $n = \varepsilon \cdot 2$, $\varepsilon = \pm 1$. Пусть для всех целых чисел a ($a \neq -1, 0, 1$), $|a| < |n|$, доказано существование нужного представления. Покажем, что для n такое представление тоже возможно. Если $|n|$ – простое число, то, обозначив его $|n| = p_1$, получим $n = \pm|n| = \pm \varepsilon p_1$, где $\varepsilon = \pm 1$.

Пусть теперь число n составное, $|n| > 1$, т. е. существует нетривиальный делитель a ($a \neq \pm 1$, $a \neq \pm n$) числа n. Тогда существует такое целое число b, что $n = ab$, где $b \neq \pm 1$, $b \neq \pm n$. Поскольку $|n| > 1$, справедливо равенство для абсолютных величин: $|n| = |ab| = |a| \cdot |b| > |a|$.

Аналогично $|n| > |b|$, поэтому к числам a и b применимо предположение индукции: $a = \varepsilon_1 p_1 p_2 \ldots p_k$, $b = \varepsilon_2 q_1 q_2 \ldots q_l$, где числа p_i , q_j простые, ε_1, $\varepsilon_2 = \pm 1$. Произведение $a \cdot b$ дает требуемое представление. Существование представления доказано.

Теперь покажем единственность. Пусть $n = \varepsilon_1 p_1 p_2 \ldots p_k = \varepsilon_2 q_1 q_2 \ldots q_l$, где числа p_i, q_j простые, $\varepsilon_1 = \pm 1$, $\varepsilon_2 = \pm 1$. Тогда $\varepsilon_1 = \varepsilon_2 = -1$ при $n < 0$. Далее, $p_1 p_2 \ldots p_k$, $= q_1 q_2 \ldots q_l$, тогда по свойству 5 простых чисел $k = 1$ и числа $q_1 q_2 \ldots q_l$ можно перенумеровать так, что $p_1 = q_1$, $p_2 = q_2, \ldots, p_k = q_l$.

Представление числа n в виде

$$n = p_1^{\alpha 1} p_2^{\alpha 2} \ldots p_s^{\alpha s},$$

где $\varepsilon = \pm 1$, $p_1, p_2, \ldots, p_s$ – различные простые числа, $\alpha_i \geq 1$ для $i = 1, 2, \ldots, s$, $s \geq 1$, называется каноническим разложением числа n.

ВЫЧИСЛЕНИЕ НАИБОЛЬШЕГО ОБЩЕГО ДЕЛИТЕЛЯ

Алгоритм Евклида

При работе с большими составными числами их разложение на простые множители, как правило, неизвестно. Но для многих прикладных задач теории чисел поиск разложения числа на множители является важной, часто встречающейся практической задачей. В теории чисел существует сравнительно быстрый способ вычисления НОД двух чисел, который называется алгоритмом Евклида.

Алгоритм 1. Алгоритм Евклида [3].

Вход. Целые числа a, b; $0 < b < a$.

Выход. d = НОД (a,b).

1. Положить $r_0 \leftarrow a$, $r_1 \leftarrow b$, $i \leftarrow 1$.
2. Найти остаток r_{i+1} от деления r_{i-1} на r_i.
3. Если $r_{i+1} = 0$, то положить $d \leftarrow r_i$. В противном случае положить $i \leftarrow i + 1$ и вернуться на шаг 2.
4. Результат: d.

Теорема. Для любых a, $b > 0$ алгоритм Евклида останавливается и выдаваемое им число d является наибольшим общим делителем чисел a и b.

Доказательство [3]. По теореме о делении с остатком для любого $i \geq 1$ имеем $r_{i-1} = q_i r_i + r_{i+1}$, где $0 \leq r_{i+1} < r_i$. Получаем монотонно убывающую последовательность неотрицательных целых чисел $r_1 > r_2 > r_3 > ... \geq 0$, ограниченную снизу. Такая последовательность не может быть бесконечной, значит, алгоритм Евклида останавливается.

Бинарный алгоритм Евклида

Бинарный алгоритм Евклида вычисления НОД оказывается более быстрым [3] при реализации этого алгоритма на компьютере, поскольку использует двоичное представление чисел a и b. Бинарный алгоритм Евклида основан на следующих свойствах наибольшего общего делителя (считаем, что $0 < b \leq a$):

1) если оба числа a и b четные, то НОД(a,b) = 2 · НОД(a/2 , b/2)
2) если число a нечетное, число b четное, то НОД(a, b) = НОД(a, b/2);
3) если оба числа a и b нечетные, $a > b$, то НОД(a, b) = НОД($a – b$, b);
4) если $a = b$, то НОД(a, b) = a.

Алгоритм 2. Бинарный алгоритм Евклида [3].

Вход. Целые числа a, b; $0 < b \leq a$.

Выход. d = НОД(a,b).

1. Положить $g \leftarrow 1$.

2. Пока оба числа a и b четные, выполнять $a \leftarrow a/2$, $b \leftarrow b/2$, $g \leftarrow 2g$ до получения хотя бы одного нечетного значения a или b.

3. Положить $u \leftarrow a$, $v \leftarrow b$.

4. Пока $u \neq 0$, выполнять следующие действия.

4.1. Пока u четное, полагать $u \leftarrow u/2$.

4.2. Пока v четное, полагать $v \leftarrow v/2$.

4.3. При $u \geq v$ положить $u \leftarrow u - v$.

В противном случае положить $v \leftarrow v - u$.

5. Положить $d \leftarrow gv$.

6. Результат: d.

Расширенный алгоритм Евклида

Расширенный алгоритм Евклида [3] находит наибольший общий делитель d чисел a и b и его линейное представление, т. е. целые числа x и y, для которых $ax + by = d$, и не требует «возврата», как в рассмотренном примере.

Пусть d – НОД для a и b, т. е. $d = (a, b)$, где $a > b$. Тогда существуют такие целые числа x и y, что $d = ax + by$. Иными словам, НОД двух чисел можно представить в виде линейной комбинации этих чисел с целыми коэффициентами.

Алгоритм 3. Схема расширенного алгоритма Евклида.

1. Определить $a_0 = 1$, $a_1 = 0$, $b_0 = 0$, $b_1 = 1$, $\alpha = a$, $\beta = b$.

2. Пусть число q – частное от деления числа a на число b, a число r – остаток от деления этих чисел (т. е. $a = qb + r$).

3. Если остаток от деления r равен нулю, то выполняем шаг 6.

4. Определяем:

$a = b$;

$b = r$;

$t = a_0$; // $t = x_{i-1}$

$a_0 = a_1$;

$a_1 = t - a_1 q$; // $a_1 = x_i$ – для правой части = x_{i+1} для правой

$t = b_0$; // $t = y_{i-1}$

$b_0 = b_1$;

$b_1 = t - b_1 q$;

5. Возвращаемся на шаг 2.

6. Определяем $x = x_0$, $y = y_0$, $d = \alpha x + \beta y$.

Вариант расширенного алгоритма Евклида

Вход. Целые числа a, b; $0 < b \leq a$.

Выход: d = НОД(a, b); такие целые числа x, y, что $ax + by = d$.

1. Положить $r_0 \leftarrow a$, $r_1 \leftarrow b$, $x_0 \leftarrow 1$, $x_1 \leftarrow 0$, $y_0 \leftarrow 0$, $y_1 \leftarrow 1$, $i \leftarrow 1$

2. Разделить с остатком r_{i-1} на r_i,: $r_{i-1} = q_i\, r_i + r_i + 1$

3. Если $r_{i+1} = 0$, то положить $d \leftarrow r_i$, $x \leftarrow x_i$ $y \leftarrow y_i$. В противном случае положить $x_{i+1} \leftarrow x_{i-1} - q_i x_i$, $y_{i+1} \leftarrow y_{i-1} - q_i y_i$, $i \leftarrow i + 1$ и вернуться на шаг 2.

4. Результат: d, x, y.

Корректность определения чисел x и y, вычисляемых алгоритмом, показывает следующая теорема.

Теорема 4. На каждой итерации алгоритма 3 выполняется равенство $ax_i + by_i = r_i$, при $i \geq 0$.

Доказательство [3]. Воспользуемся методом математической индукции. Для $i = 0$ и $i = 1$ требуемое равенство имеет место в силу шага 1 алгоритма 3. Допустим, что оно справедливо для $i - 1$ и для i. Тогда на шаге 3 получаем

$$x_{i+1} = x_{i-1} - q_i x_i \text{ и } y_{i+1} = y_{i-1} - q_i y_i.$$

Следовательно,

$$ax_{i+1} + by_{i+1} = a(x_{i-1} - q_i x_i) + b(y_{i-1} - q_i y_i,) =$$

$$= ax_{i-1} + by_{i-1} - q_i(ax_i + by_i) = r_{i-1} - q_i r_i = r_{i+1}.$$

Пример. Дано $a = 1769$, $b = 551$. Используя расширенный алгоритм Эвклида, найти целые числа x и y такие, что $d = ax + by$, где d – НОД чисел a и b.

I этап последовательности вычислений

1. Определить $a_0 = 1$, $a_1 = 0$, $b_0 = 0$, $b_1 = 1$, $\alpha = 1769$, $\beta = 551$.
2. Частное от деления $q = a/b = 1769/551 = 3$, а остаток от деления $r = 116$.
3. Если остаток от деления r равен нулю, то выполняем шаг 6.
4. Определяем:

   $a = 551$;
   $b = 116$;
   $t = a_0 = 1$;
   $a_0 = a_1 = 0$:
   $a_1 = t - a_1 q = 1 - 0 = 1$
   $t = b_0 = 0$;
   $b_0 = b_1 = 1$;
   $b_1 = t - b_1 q = -3$;

В результате текущего шага получили следующие промежуточные значения параметров:

$a = 551$, $b = 116$, $a_0 = 0$, $a_1 = 1$, $b_0 = 1$, $b_1 = -3$.

5. Так как остаток от деления $r \neq 0$, то возвращаемся на шаг 2.

II этап последовательности вычислений

1. Значение параметров:
   $a = 551$, $b = 116$, $a_0 = 0$, $a_1 = 1$, $b_0 = 1$, $b_1 = -3$.
2. Частное от деления $q = a/b = 551/116 = 4$, а остаток от деления $r = 87$.
3. Если остаток от деления r равен нулю, то выполняем шаг 6.
4. Определяем:
   $a = 116$;
   $b = 87$;
   $t = a_0 = 0$;
   $a_0 = a_1 = 1$:
   $a_1 = t - a_1 q = 0 - 1 \cdot 4 = -4$;
   $t = b_0 = 1$;
   $b_0 = b_1 = -3$;
   $b_1 = t - b_1 q = 1 - (-3) \cdot 4 = 13$;

В результате текущего шага получили следующие промежуточные значения параметров:

$a = 116$, $b = 87$, $a_0 = 1$, $a_1 = -4$, $b_0 = -3$, $b_1 = 13$.

5. Так как остаток от деления $r \neq 0$, то возвращаемся на шаг 2.

III этап последовательности вычислений

1. Значение параметров:
   $a = 116$, $b = 87$, $a_0 = 1$, $a_1 = -4$, $b_0 = -3$, $b_1 = 13$.
2. Частное от деления $q = a/b = 116/87 = 1$, а остаток от деления $r = 29$.
3. Если остаток от деления r равен нулю, то выполняем шаг 6.
4. Определяем:
   $a = 87$;
   $b = 29$;
   $t = a_0 = 1$;
   $a_0 = a_1 = -4$:
   $a_1 = t - a_1 q = 1 - (-4) \cdot 1 = 5$;
   $t = b_0 = -3$;
   $b_0 = b_1 = 13$;
   $b_1 = t - b_1 q = -3 - (13) \cdot 1 = -16$;

В результате текущего шага получили следующие промежуточные значения параметров:

$a = 87$, $b = 29$, $a_0 = -4$, $a_1 = 5$, $b_0 = 13$, $b_1 = -16$.

5. Так как остаток от деления $r \neq 0$, то возвращаемся на шаг 2.

IV этап последовательности вычислений.

1. Значение параметров:
   $a = 87$, $b = 29$, $a_0 = -4$, $a_1 = 5$, $b_0 = 13$, $b_1 = -16$.

2. Частное от деления $q = a/b = 87/29 = 3$, а остаток от деления $r = 0$.

3. Если остаток от деления r равен нулю, то выполняем шаг 6.

4. Определяем:

$a = 87$;

$b = 29$;

$t = a_0 = -4$;

$a_0 = a_1 = 5$; $-4 - 5 \cdot 3 = -19$;

$t = b_0 = 13$;

$b_0 = b_1 = -16$;

$b_1 = t - b_1 q = 13 - (-16) \cdot 3 = 61$;

В результате текущего шага получили следующие промежуточные значения параметров:

$a = 87$, $b = 29$, $a_0 = 5$, $a_1 = -19$, $b_0 = -16$, $b_1 = 61$.

5. Так как остаток от деления $r = 0$, то выполняем шаг 6.

6. Вычисляем НОД по формуле $d = \alpha x + \beta y$, где

$x = x_0 = 5$, $y = y_0 = -16$, $\alpha = 1769$, $\beta = 551$.

Подставляя значение параметров, получаем

$d = \alpha x + \beta y = 1769 \cdot 5 - 551 \cdot 16 = 8845 - 8816 = 29$.

Расширенный алгоритм Евклида можно реализовать и в двоичном виде.

Алгоритм 4. Расширенный бинарный алгоритм Евклида [3].

Вход. Целые числа a, b; $0 < b \le a$.

Выход. $d = $ НОД(a, b); такие целые числа x, y, что $ax + by = d$.

1. Положить $g \leftarrow 1$.

2. Пока оба числа a и b четные, выполнять $a \leftarrow a/2$, $b \leftarrow b/2$, $g \leftarrow 2g$ до получения хотя бы одного нечетного значения a или b.

3. Положить $u \leftarrow a$, $v \leftarrow b$, $A \leftarrow 1$, $B \leftarrow 0$, $C \leftarrow 0$, $D \leftarrow 1$.

4. Пока $u \neq 0$, выполнять следующие действия:

4.1. Пока u четное, положить $u \leftarrow u/2$.

4.1.2. Если оба числа A и B четные, то положить $A \leftarrow A/2$, $B \leftarrow B/2$.

В противном случае положить $A \leftarrow (A+b)/2$, $B \leftarrow (B-a)/2$

4.2. Пока v четное:

4.2.1. Положить $v \leftarrow v/2$.

4.2.2. Если оба числа C и D четные, то положить $C \leftarrow C/2$, $D \leftarrow D/2$.

В противном случае положить $C \leftarrow (C + b)/2$, $D \leftarrow (D - a)/2$

4.3. При $u \ge v$ положить $u \leftarrow u - v$, $A \leftarrow A - C$, $B \leftarrow B - D$.

В противном случае положить $v \leftarrow v - u$, $C \leftarrow C - A$, $D \leftarrow D - B$.

5. Положить $d \leftarrow gv$, $x \leftarrow C$, $y \leftarrow D$.

6. Результат: d, x, y.

ОТНОШЕНИЕ СРАВНИМОСТИ

Возьмем натуральное целое число m, которое будем называть модулем.

Определение. Целые числа a и b называются сравнимыми по модулю m, если разность $(a - b)$ делится на m ($m \mid a - b$).

Это соотношение между a, b и m для краткости записывается как $a \equiv b \bmod m$, где a и b будем называть соответственно левой и правой частями сравнения. Из определения 1 следует, что если a делится на m, то $a \equiv 0 \bmod m$.

Теорема 5. Число a сравнимо с b тогда и только тогда, когда a и b имеют одинаковые остатки при делении на модуль m.

Доказательство. Пусть $a \equiv b \ mod \ m$. Представим a и b в виде

$$a = mq^1 + r^1,\ b = mq^2 + r^2,$$

где $0 \le r^1 < m$, $0 \le r^2 < m$. Из этих представлений получаем

$$a - b = m(q^1 - q^2) + r^1 - r^2.$$

Так как $(m \mid a - b)$, то m должно делить $(r^1 - r^2)$. Однако имеет место – $m < r^1 - r^2 < m$. Отсюда следует, что $r^1 - r^2 = 0$, т. е. $r^1 = r^2$.

Обратно. Пусть остатки от деления a и b на m равны, т. е. имеет место $a = mq^1 + r$, $b = mq^2 + r$. Тогда $a - b = m(q^1 - q^2)$, т. е. $m \mid a - b$.

На основании теоремы можно дать следующее определение сравнимости чисел a и b.

Определение. Целые числа a и b называются сравнимыми по модулю m, если остатки от деления этих чисел на m равны.

Свойства сравнений

Теорема. Отношения сравнимости рефлексивно, т. е. $a \equiv a \bmod m$.

Доказательство следует из того, что левая и правая части сравнения имеют равные остатки.

Теорема 6. Отношения сравнимости симметрично, т. е. если $a \equiv b \bmod m$, то $b \equiv a \bmod m$.

Доказательство. Если a и b имеют одинаковые остатки при делении на m, то остатки от деления b и a на m также равны.

Теорема 7. Отношения сравнимости транзитивно, т. е. если $a \equiv b \bmod m$, $b \equiv c \bmod m$, то $a \equiv c \bmod m$.

Доказательство. Если у чисел a и b, а также у чисел b и c остатки от деления на m равны, то a и c имеют одинаковые остатки при делении на m.

Теорема 8. Если $a \equiv b \bmod m$ и k произвольное целое число, то $ka \equiv kb \bmod m$.

Доказательство. Если $a \equiv b \bmod m$, то $m|a - b$. Тогда $m|k(a - b)$ или $m|ka - kb$. Следовательно, $ka \equiv kb \bmod m$.

Определение. Пусть a и b целые числа, причем $b \neq 0$. Число b называется делителем a, если существует целое число k такое, что $a = kb$.

Определение. Общим делителем двух чисел a и b называется любое целое k, такое, что $k|a$ и $k|b$.

Определение. Наибольшим общим делителем (НОД) двух чисел a и b называется такой положительный общий делитель k, который делится на любой другой общий делитель этих чисел. Наибольший общий делитель чисел a и b обозначается следующим образом: $(a, b) = k$.

Определение. Числа a и b называются взаимно простыми, если наибольший общий делитель этих чисел равен 1, т. е. $(a,b) = 1$.

Теорема 9. Если $ka \equiv kb \bmod m$ и $(k,m) = 1$, то $a \equiv b \bmod m$.

Доказательство. Если $ka \equiv kb \bmod m$, то $m|ka - kb$, т. е. $m|k(a - b)$. Так как по условию вышеприведенных теорем наибольший общий делитель равен 1, то из условия $m|k(a - b)$ следует, что $m|a - b$.

Теорема. Если $a \equiv b \bmod m$ и k – произвольное целое натуральное число, то $ka \equiv kb \bmod km$.

Доказательство. Если $m|a - b$, то $km|ka - kb$. Отсюда следует, что $ka \equiv kb \bmod km$.

Теорема 10. Если $a \equiv b \bmod m$, $c \equiv d \bmod m$, то

$$a + c \equiv (b + d) \bmod m,\ a - c \equiv (b - d) \bmod m.$$

Доказательство. Если $m|a - b$ и $m|c - d$, то $m|(a - b) \pm (c - d)$. Отсюда получаем $m|(a \pm c) - (b \pm d)$.

Теорема. Если $a \equiv b \bmod m$, $c \equiv d \bmod m$, то $ac \equiv bd \bmod m$.

Доказательство. Если $a \equiv b \bmod m$ и $c \equiv d \bmod m$, то имеем

$$ac \equiv bc \bmod m \text{ и } cb \equiv db \bmod m.$$

Тогда из свойства транзитивности сравнений получим, что $ac \equiv bd \bmod m$.

Теорема 11. Если $a \equiv b \bmod m$, то при любом целом $n \geq 0$ имеет место $an \equiv bn \bmod m$.

Доказательство. Применяя n раз теорему 9 для сравнений $a \equiv b \bmod m$ и $a \equiv b \bmod m$, получим доказательство.

Пусть n – любое положительное целое число, $p(x)$ – многочлен степени n с целыми коэффициентами. Тогда из сформулированных выше теорем вытекает следующая теорема.

Теорема. Если $a \equiv b \bmod m$, то $p(a) \equiv p(b) \bmod m$.

МОДУЛЯРНАЯ АРИФМЕТИКА

Модулярная арифметика или арифметика остатков является основой современной криптографии. Эта арифметика заключается в том, что у любого выражения, в котором используются только целые числа, требуется вычислить не значение этого выражения, а остаток от деления этого значения на некоторое целое число n.

Арифметика остатков во многом совпадает с обычной арифметикой. Она коммутативна, ассоциативна и дистрибутивна. Кроме того, вычисление промежуточного результата по модулю n дает такой же результат, что и выполнение всего вычисления с последующим вычислением остатка, т. е. справедливы формулы

$$(a + b) \bmod n \equiv ((a \bmod n) + (b \bmod n)) \bmod n$$

$$(a - b) \bmod n \equiv ((a \bmod n) - (b \bmod n)) \bmod n$$

$$(a \cdot b) \bmod n \equiv ((a \bmod n) \cdot (b \bmod n)) \bmod n$$

$$(c(a + b)) \bmod n \equiv (((ca) \bmod n)+((cb) \bmod n)) \bmod n.$$

Пока при составлении арифметических выражений можно использовать только операции сложения, умножения и вычитания. В дальнейшем мы научимся вычислять обратный элемент, логарифм числа и квадратный корень по модулю n.

Вычисление $a^x \bmod n$

В случае, когда показатель степени x является степенью 2, для вычисления $a^x \bmod n$ можно использовать формулу, которая для $x = 16$ имеет следующий вид:

$$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n.$$

Если показатель x не является степенью 2, то, записав этот показатель в двоичной системе счисления, можно представить x в виде суммы степеней 2, что позволяет значительно уменьшить число операций умножений.

КЛАССЫ

Определение. Классом вычетов по модулю n называется множество всех целых чисел, сравнимых с некоторым данным целым числом a. Такой класс обозначается через $[a]n$.

Условимся, что если значение модуля однозначно определенно, то индекс в обозначении класса будем опускать. Множество всех классов вычетов по модулю n обозначим через Z/n. Заметим, что в определении не исключается случай, когда число a больше модуля n.

Теорема 12. Класс чисел, сравнимых с a по модулю n, совпадает с множеством чисел вида $a + nk$, где k – произвольное целое число.

Доказательство. Если целое число $x \in [a]$, то $x \equiv a \bmod n$, т. е. $x - a = nk$ или $x = a + nk$, где k – любое целое число. В то же время при любом значении k имеем $a + nk \equiv a \bmod n$. Откуда ясно, что множество чисел $a + nk$ совпадает с классом $[a]$.

Из теоремы 12 следует, что каждый класс содержит бесконечно множество чисел.

Пример. Пусть модуль $n = 7$. В этом случае класс [9] состоит из чисел …, –12, –5, 2, 9, 16, 23, …,

Теорема 13. $[a] = [b]$ тогда и только тогда, когда $a \equiv b \bmod n$.

Доказательство. Если $a \equiv b \bmod n$, то для любого $x \in [a]$ следует соотношение $x \equiv a \bmod n$.

Из транзитивности сравнения вытекает, что $x \equiv b \bmod n$,т. е. $x \in [b]$. В силу симметрии a и b следует, что для любого $x \in [b]$ имеет место $x \in [a]$. Таким образом, классы $[a]$ и $[b]$ совпадают.

Обратно. Если $[a] = [b]$, то это означает, что $a \equiv b \bmod n$.

Теорема 14. Если два класса имеют хотя бы один общий элемент, то они совпадают.

Доказательство. Если $x \in [a]$ и $x \in [b]$, то $x \equiv a \bmod n$, $x \equiv b \bmod n$, то по свойству транзитивности $a \equiv b \bmod n$.

Тогда из теоремы 13 вытекает, что $[a] = [b]$

Теорема 15. Если какое-то число класса по модулю n имеет при делении на n остаток равный r, то все числа класса имеют при делении на n остаток r.

Доказательство. Пусть $x = nk + r$, $0 \le r < n$.

Тогда по теореме 12 все числа из класса, которому принадлежит x, имеют вид

$$x + nt = n(k + t) + r = ns + r,$$

где $s = k + t$ – любое целое число.

Теорема 16. Число классов по модулю n конечно и равно n.

Доказательство. По теореме 15 все числа одного класса имеют при делении на n один и тот же остаток. Поэтому классы по модулю n взаимно однозначно соответствуют остаткам. Остатками при делении на n являются числа 0, 1, 2, …, $n - 1$.

Определение 7. Вычетом класса называется любое из чисел, которое принадлежит данному классу.

Среди неотрицательных (положительных) вычетов класса обязательно есть наименьшее, которое называется наименьшим положительным вычетом класса.

Теорема 17. Наименьший неотрицательный вычет класса $[a]$ по модулю n равен остатку от деления a на n.

Доказательство. Пусть r – остаток от деления a на n, т. е. $a = nq + r$.

Тогда $a \equiv r \ mod \ n$ и $[a] = [r]$. В этом случае, если $c \in [a]$, то $c = r + nt$.

Если $t < 0$, то $r + nt \leq r - m < 0$. В результате имеем, что $c > 0$, если $t = 0$.

Теорема 18. Множество классов по данному модулю представляют собой аддитивную группу, т. е. множество с одной операцией сложения, нулевым и противоположным элементом.

Теорема 19. Множество классов по данному модулю представляет собой коммутативное кольцо.

Полная и приведенная система вычетов

Определение. Полной системой вычетов по некоторому модулю называется система чисел, взятых по одному из каждого класса по этому модулю.

Если из каждого класса по некоторому модулю взять любое число, то получается полная система вычетов.

Пример. Числа 12, –23, 2, 63, –2, 5 образуют полную систему вычетов по модулю 6.

Замечание. Остаток от деления –23 на 6 равен –5 по правилам обычной арифметики. Чтобы определить класс, которому принадлежит число –23, надо к остатку –5 прибавить модуль 6. Получим, что число –23 принадлежит классу, который соответствует остатку 1. Аналогичный результат получаем, если при делении будем добиваться положительного остатка. В этом случае частное от деления –23 на 6 будет равен –4, а остаток равен 1.

Обычно в качестве представителей классов берут наименьшее неотрицательное число из каждого класса. Отсюда следует, что системой наименьших неотрицательных вычетов по модулю n являются следующие числа: 1, 2, 3, …, n.

Теорема 20. Если $(a, n) = 1$, т. е. a и n – взаимно просты, b – произвольное целое число, а x пробегает полную систему вычетов по модулю n,

то числа $ax + b$ также принимают значения, которые образуют полную систему вычетов по этому модулю.

Определение. Приведенной системой вычетов по некоторому модулю называется система чисел, которые взяты по одному из каждого класса, взаимно простого с модулем.

Функция Эйлера

Определение. Число классов по модулю n, взаимно простых с этим модулем, называется функцией Эйлера $\varphi(n)$.

Так как число классов, взаимно простых с модулем n, равно числу целых чисел, не превосходящих n и взаимно простых с n, то можно дать следующее эквивалентное определение функции Эйлера.

Определение. Число натуральных чисел, взаимно простых с n и не превосходящих n, называется функцией Эйлера $\varphi(n)$.

Пример. Для модуля $n = 1$ значение функции Эйлера $\varphi(1) = 1$. Для модуля $n = 12$ в приведенной системе вычетов 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 только четыре числа 1, 5, 7, 11 являются взаимно простыми с модулем 12. Поэтому $\varphi(12) = 4$.

Определение. Функция $f(n)$, определенная на множестве натуральных чисел, называется мультипликативной, если для любых взаимно простых натуральных чисел a и b имеет место $f(a, b) = f(a)f(b)$.

Теорема 21. Функция Эйлера мультипликативная, т. е. имеет место соотношение $\varphi(ab) = \varphi(a)\varphi(b)$, если числа a и b взаимно просты, т. е. наибольший общий делитель равен единице или $(a, b) = 1$.

Теорема 22. Пусть p – простое число и α – любое натуральное, тогда $\varphi(p^{\alpha}) = p^{\alpha-1}(p - 1)$. Заметим, что для простого p функция Эйлера $\varphi(p) = p - 1$.

Теорема 23. Пусть $m = p_1^{\alpha 1} p_2^{\alpha 2} \ldots p_k^{\alpha k}$, где p_i – простые числа и α_i, $i \le k$, – любые натуральные числа. Тогда

$$\varphi(p_1^{\alpha 1} p_2^{\alpha 2} \ldots p_k^{\alpha k}) = p_1^{\alpha 1-1}(p_1 - 1)\, p_2^{\alpha 2-1}(p_1 - 1) \ldots p_1^{\alpha k-1}(p_k - 1).$$

Примеры. Пусть для модуля $n = 2$ и $n = 3$ соответственно $\varphi(2) = 1$, $\varphi(3) = 2$. Тогда $\varphi(6) = \varphi(2 \times 3) = \varphi(2)\varphi(3) = 2$.

В тоже время $\varphi(18) = \varphi(3 \times 6) \ne \varphi(3)\varphi(6)$, потому что числа 3 и 6 не взаимно просты, наибольший общий делитель этих чисел равен трем (3,6) = 3. Однако имеет место

$$\varphi(18) = \varphi(2 \times 9) = \varphi(2 \times 32) = (2 - 1)\, 32 - 1(3 - 1) = 3 \times 2 = 6.$$

Теорема (Ферма). Для любого простого p и любого a, $a \ge 1$, которое не делится на p, справедливо сравнение $a^{p-1} \equiv 1 \bmod p$, или $a^p \equiv a \bmod p$.

Заметим, что $p - 1$ для простого p совпадет с функцией Эйлера $\varphi(p) = p - 1$. Поэтому доказываемое сравнение теоремы Ферма можно записать в виде $a^{p-1} = a^{\varphi(p)} \equiv 1 \bmod p$. Оказывается, что последнее сравнение справедливо для любого модуля m, взаимно простого с числом a, $a \geq 1$. Последний факт подтверждается следующей теоремой Эйлера.

Теорема (Эйлера). Для любого модуля m и любого $a \geq 1$, взаимно простого с m, справедливо сравнение $a^{\varphi(m)} \equiv 1 \bmod m$.

Теоремы Ферма и Эйлера позволяют находить остатки от деления на модуль больших степеней заданного числа. Если надо найти остаток от деления an на модуль m, где $(a, m) = 1$, т. е. общий делитель a и m равен 1, и $\varphi(m) < n$, то число n можно представить в виде $n = q\varphi(m) + r$, где $0 \leq r < \varphi(m)$. Тогда $a^n = a^{q\varphi(m)+r} = (a^{\varphi(m)})^q a^r \equiv a^r \bmod m$.

Если у чисел a и m существует наибольший общий делитель d, т. е. $(m, a) = d$, то $m = m_1 d$ и $a = a_1 d$, где $(m_1, a_1) = 1$. Обозначим через r остаток от деления a^n на m. Тогда имеет место

$$a^n = a^{n-1}a = a^{n-1}a_1 d \equiv r \bmod m \equiv r \bmod m_1 d.$$

По свойству сравнений можно написать $r = r_1 d$. Поэтому $a^{n-1}a_1 d \equiv r \bmod m \equiv r_1 d \bmod m_1 d$ или $a^{n-1}a_1 \equiv r_1 \bmod m_1$. Вычисления остатка r_1 от деления an на m сводится к вычислению остатков от деления чисел a_1 и a^{n-1} на модуль m_1.

Пример. Найти остаток от деления 171^{2147} на 52. Для определения остатка от деления выполним следующие шаги.

1. Определяем значение функции Эйлера для модуля 52, т. е.

$$\varphi(52) = \varphi(4 \times 13) = \varphi(2^2 \times 13) = 2^{2-1}(2-1)\times(13-1) = 2\times 12 = 24.$$

2. Определяем результат деления степени 2147 на значение функции Эйлера от модуля 52, имеем $\varphi(52) = 24$. Частное от деления q равно 89, а остаток r равен 11. Поэтому можно записать

$$2147 = q \cdot \varphi(52) + r = 89 \cdot 24 + 11.$$

3. Имеем

$$171^{89\times 24+11} = 171^{89\times 24}171^{11} = (171^{24})^{89}171^{11} = \\ = (171^{\varphi(52)})^{89}171^{11}.$$

По теореме Эйлера $171^{\varphi(52)} = 1$. По свойству сравнений, если основание степени $a > m$, m – модуль, и $a = qm + r$, где q – частное, а r – остаток от деления a на m, то $a^n \bmod m \equiv (qm + r)^n \bmod m \equiv r^n \bmod m$. Так как осно-

вание степени 171 больше модуля 52, то можно записать $171 = 3 \times 52 + 15$, где 3 – частное от деления числа 171 на модуль 52, а 15 – остаток. Учитывая проведенные выкладки, получаем

$$(171^{\varphi(52)})^{89}171^{11} \bmod 52 \equiv (1)891\ 511 \bmod 52 \equiv 15^{11} \bmod 52 \equiv$$
$$\equiv 15^8 15^3 \bmod 52.$$

Используя формулу для возведения в степень

$$a^8 \bmod n = ((a^2 \bmod n)^2 \bmod n)^2 \bmod n,$$

получаем

$$15^2 \bmod 52 \equiv 225 \bmod 52 \equiv 17 \bmod 52.$$

Далее имеем

$$17^2 \bmod 52 \equiv 289 \bmod 52 \equiv 29 \bmod 52.$$

Теперь можно записать

$$15^8 \bmod 52 \equiv ((15^2 \bmod 52)^2 \bmod 52)^2 \bmod 52 \equiv$$
$$\equiv 29^2 \bmod 52 \equiv 841 \bmod 52.$$

Окончательно получим

$$15^8 \bmod 52 \equiv 841 \bmod 52 \equiv 9 \bmod 52.$$

По свойству сравнений имеем

$$15^8 15^3 \bmod 52 \equiv$$
$$\equiv (15^8 \bmod 52)\ (15^2 \bmod 52)\ (15 \bmod 52) \equiv (9 \times 17 \times 15) \bmod 52.$$

Далее запишем

$$(9 \times 17 \times 15) \bmod 52 \equiv (9 \times 17 \bmod 52) \times 15 \bmod 52 \equiv$$
$$\equiv (153 \bmod 52) \times 15 \bmod 52 \equiv (49 \bmod 52) \times (15 \bmod 52) \equiv$$
$$\equiv (49 \times 15) \bmod 52 \equiv 735 \bmod 52 \equiv 7 \bmod 52.$$

Окончательно получим

$$171^{2147} \equiv (171^{\varphi(52)})^{89}171^{11} \bmod 52 \equiv 171^{11} \bmod 52 \equiv$$
$$\equiv 15^{11} \bmod 52 \equiv 7 \bmod 52.$$

Задача вычисления остатка от деления 171^{2147} на 52 завершена.

Пример. Найти остаток от деления 126^{1020} на 138. Заметим, что основание степени 126 и модуль 138 имеет наибольший общий делитель, равный 6. Для определения остатка от деления выполним следующие шаги.

1. Преобразуем степень и модуль

$$126^{1020} = 126^{1019} \times 126 = 126^{1019} \times 21 \times 6;\ 138 = 23 \times 6.$$

Тогда

$$126^{1020} \bmod 138 \equiv 126^{1019} \times 21 \times 6 \bmod (23 \times 6) \equiv$$
$$\equiv 126^{1019} \times 21 \bmod 23.$$

2. Определяем значение функции Эйлера для модуля 23. Так как 23 есть простое число, то $\varphi(23) = 23 - 1 = 22$;

3. Определяем результат деления степени 1019 на значение функции Эйлера от модуля 23, т. е. $\varphi(23) = 22$. Частное от деления равно q=46, а остаток r равен 7. Поэтому можно записать $1019 = q \times \varphi(23) + r = 22 \times 46 + 7$;

4. Имеет место соотношение

$$126^{1019} \times 21 \bmod 23 \equiv$$
$$\equiv (126^{22 \times 46+7} \times 21) \bmod 23 \equiv ((126^{22})^{46}\, 126^{7} \times 21) \bmod 23.$$

Так как по теореме Эйлера можно записать

$$126^{\varphi(23)} \bmod 23 \equiv 126^{22} \bmod 23 \equiv 1 \bmod 23,$$

то получаем

$$126^{1020} \bmod 138 \equiv 126^{7} \times 21 \bmod 23.$$

Учитывая, что остаток от деления числа 126 на модуль 23 равен 11, то по свойствам сравнений можно записать

$$126^{7} \times 21 \bmod 23 \equiv (126 \bmod 23)^{7} \times 21 \bmod 23 \equiv$$
$$\equiv 11^{7} \bmod 23 \times 21 \bmod 23.$$

Вычислим $11^{7} \bmod 23$. Можно записать

$$11^{7} \bmod 23 \equiv 11^{6+1} \bmod 23 \equiv (11^{6} \times 11) \bmod 23 \equiv$$
$$\equiv 11^{6} \bmod 23 \times 11 \bmod 23.$$

Далее имеем

$$11^{6} \bmod 23 \equiv (11^{2} \bmod 23)^{3} \bmod 23 \equiv (121 \bmod 23)^{3} \bmod 23.$$

Учитывая, что остаток от деления числа 121 на модуль 23 равен 6, получаем

$$(121 \bmod 23)^3 \bmod 23 \equiv (6 \bmod 23)^3 \bmod 23.$$

Используя свойства сравнений, имеем

$$(6 \bmod 23)^3 \bmod 23 \equiv 6^3 \bmod 23 \equiv 216 \bmod 23 \equiv 9 \bmod 23.$$

Итак, в результате запишем

$$11^7 \bmod 23 \equiv 9 \bmod 23 \times 11 \bmod 23 \equiv$$
$$\equiv (9 \times 11) \bmod 23 \equiv 7 \bmod 23.$$

Далее получаем

$$126^{1020} \bmod 138 \equiv 7 \bmod 23 \times 21 \bmod 23 \equiv (7 \times 21) \bmod 23 \equiv$$
$$\equiv 147 \bmod 23 \equiv 9 \bmod 23.$$

Окончательно имеем

$$126^{1020} \bmod 138 \equiv 6 \times 1 \,(26^{1019} \times 21 \bmod 23) \equiv$$
$$\equiv 6 \times 9 \bmod 138 \equiv 54 \bmod 138.$$

Таким образом, остаток от деления числа 126^{1020} на 138 равен 54. Задача вычисления остатка от деления 126^{1020} на 138 завершена.

Результаты Ферма и Эйлера, которые были получены соответственно в 1640 и 1760 году, стали фундаментом для формирования криптографической системы *RSA* с открытым ключом. В 1977 году трое американских ученых, Р. Ривест, А. Шамир и Л. Адлеман, предложили эту принципиально новую и важную для современных информационных технологий систему. Фактически же система *RSA* полностью базируется на теореме Эйлера.

СРАВНЕНИЯ ПЕРВОЙ СТЕПЕНИ

Пусть a и b – целые числа. Обозначим через целое m модуль для вычисления сравнений. Тогда выражение

$$ax \equiv b \bmod m$$

называется сравнением первой степени с одним неизвестным. Решить сравнение – значит найти все значения x, которые удовлетворяют данному сравнению.

Теорема. Если $(a, m) = d$, т. е. число a и модуль m имеет общий делитель d, причем число d не делит b, то сравнение

$$ax \equiv b \bmod m$$

не имеет решений.

Согласно этой теореме решение сравнения $ax \equiv b \bmod m$ существует только в случае, если $d|b$. Тогда имеем $a = a_1d$, $b = b_1d$, $m = m_1d$, а исходное сравнение можно записать в виде

$$a_1dx - b_1d \bmod m_1d.$$

Используя свойство сравнения, последнее соотношение можно заменить эквивалентным

$$a_1x \equiv b_1 \bmod m_1,$$

где $(a_1, m_1) = 1$. Поэтому, не ограничивая общности, изучать сравнение

$$ax \equiv b \bmod m$$

можно, предполагая, что коэффициент при неизвестном и модуль являются взаимно простыми числами, т. е. НОД $(a, m) = 1$.

Теорема. Если $(a, m) = 1$, т. е. числа a и m взаимно просты, то сравнение

$$ax \equiv b \bmod m$$

имеет одно и только одно решение.

Теорема. Если $(a, m) = 1$, т. е. числа a и b взаимно простые, то решением сравнения

$$ax \equiv b \bmod m$$

является класс

$$x \equiv ba^{\varphi(m)-1} \bmod m,$$

где $\varphi(m)$ – значение функции Эйлера для модуля m.

Пример. Решить сравнение

$$3x \equiv 4 \bmod 34.$$

Заметим, что (3,4) = 1, т. е. числа 3 и 4 взаимно просты. Тогда решение определяется формулой

$$x \equiv 4 \times 3^{\varphi(34)-1} \bmod 34.$$

Вычислим

$$\varphi(34) = \varphi(2 \times 17) = (2 - 1) \times (17 - 1) = 16.$$

Тогда решение определяется по формуле

$$x \equiv 4 \times 3^{16-1} \equiv 4 \times 3^{15} \bmod 34.$$

Вычислим значение $3^{15} \bmod 34$, используя свойство сравнений. Имеем

$$3^{15} = (3^4)^3 \times 3^3 \bmod 34 \equiv (81 \bmod 34)^3 \times 27 \bmod 34 \equiv$$
$$\equiv (13 \bmod 34)^3 \times 27 \bmod 34 \equiv (13^3 \bmod 34) \times 27 \bmod 34 \equiv$$
$$(2197 \bmod 34) \times 27 \bmod 34 \equiv \bmod (21 \bmod 34) \times 27 \bmod 34 \equiv$$
$$(21 \times 27) \bmod 34 \equiv 567 \bmod 34 \equiv 23 \bmod 34.$$

Окончательно для решения сравнения получаем следующее значение для *x:*

$$x \equiv 4 \times 23 \bmod 34 \equiv 92 \bmod 34 \equiv 24 \bmod 34.$$

Проверка. Для проверки полученного решения

$$x = 24 \bmod 34$$

подставим x в исходное сравнение

$$3x \equiv 4 \bmod 34.$$

Тогда получим

$$(3 \times 24) \bmod 34 \equiv 72 \bmod 34 \equiv 4 \bmod 34,$$

потому что при делении числа 72 на модуль 34 частное равно 2, а остаток 4. Отсюда видим, что найденное решение x удовлетворяет данному сравнению. Задача решения сравнения $3x \equiv 4 \bmod 34$ завершена.

Если число m достаточно большое, то разложить его на простые множители, чтобы вычислить значение функции Эйлера $\varphi(m)$ по модулю m и чтобы найти решение сравнения

$$ax \equiv b \bmod m$$

по формуле

$$x \equiv ba^{\varphi(m)-1} \bmod m,$$

может оказаться сложной задачей. Поэтому обычно для определения решения сравнения используют представление НОД $(a,m) = 1$ в виде линейной комбинации. Действительно, используя расширенный алгоритм Эвклида, представляем наибольший общий делитель в виде

$$1 = az + my.$$

Умножая последнее равенство на b, получаем

$$b = abz + mby.$$

Из последнего соотношения имеем

$$b - abz = mby.$$

Разность чисел b и abz делится на m (значит эти числа сравнимы по модулю m), т. е.

$$abz \equiv b \pmod m,$$

откуда следует, что число $x = bz$ будет решением исходного сравнения

$$ax \equiv b \bmod m.$$

Напомним, что число z является коэффициентом при линейном представлении НОД(a, m).

КРИПТОГРАФИЯ С ОТКРЫТЫМ КЛЮЧОМ

Продемонстрируем, как изложенные выше факты из теории чисел используются в современной криптографии с открытым ключом. Кстати, заметим, что фактически новый подход в современной криптографии использует результаты теории чисел, которые были доказаны до 1760 года. Идея криптографии с открытыми ключами была опубликована в 1976 году в работе Диффи и Хеллмана «Новые направления в криптографии». Эта идея базируется на построении функции F, при помощи которой, зная значение аргумента x, можно эффективно вычислить

$$y = F(x).$$

В то же время по значению y невозможно за приемлемый промежуток времени решить обратную задачу, т. е. по y определить аргумент x. Функции с такими свойствами называются односторонними.

Определение. Функция F: $X \to Y$ называется односторонней, если она обладает двумя свойствами:

- существует полиномиальный алгоритм вычисления значения $y = F(x)$;
- не существует полиномиального алгоритма для решения обратной задачи, т. е. определения значения x по y.

Решение обратной задачи для односторонней функции одинаково невыполнимо как для санкционированного, так и несанкционированного решения. Для разрешения данной коллизии Диффи и Хеллман ввели понятие односторонней функции с секретом (*one-way trapdoor function*).

Определение. Функция

$$FK : X \to Y,$$

которая зависит от параметра K, называется односторонней с секретом K, если она обладает тремя свойствами

- при любом параметре K существует полиномиальный алгоритм вычисления значения

$$y = FK(x);$$

- при неизвестном K не существует полиномиального алгоритма для решения обратной задачи, т. е. определения значения x по y;
- при известном K существует полиномиальный алгоритм для решения обратной задачи, т. е. определения значения x по y.

Решение обратной задачи для односторонней функции с секретом становится выполнимой для санкционированного решения, но невыполнимо для несанкционированного решения.

Для практических целей в криптографии при построении функций, которые исполняют роль односторонних, используют сложные задачи из теории чисел, например, следующие:

- Разложить на множители число n.
- Определить, является ли число n простым.
- Найти простое число больше n.
- Найти число x из уравнения

$$x^2 \equiv a \bmod n.$$

- Найти число x из уравнения

$$ax \equiv b \bmod n.$$

В последнее время развиваются криптографические методы с открытыми ключами, которые базируются на теории эллиптических кривых. Суть криптографии с открытым ключом заключается в следующем. В криптографической системе существуют два разных ключа. Расшифровать и зашифровать может только тот, кто владеет этими двумя ключами. При таком подходе можно разрабатывать различные схемы использования открытых ключей, например, следующие:

- есть пользователь, который владеет двумя ключами;
- нет пользователей, которые владеют одновременно двумя ключами.

В зависимости от разработанной схемы можно выполнять различные криптографические процедуры: аутентификацию, электронную подпись, формирование общего ключа, криптографические протоколы. Данные процедуры в свою очередь решают различные задачи, связанные с обменом информацией и т. д. В предложенных выше схемах с открытыми ключами в качестве примера их использования будут предлагаться алгоритмы цифровой подписи и аутентификации. Определим данные понятия.

Аутентификация – процедура (функция) проверки подлинности. Аутентификация может относиться ко всем аспектам взаимодействия при передаче информации – сеансу связи, передаваемому сообщению, источнику сообщения и т. д.

Электронная цифровая подпись (**ЭЦП**) – процедура, которая обеспечивает невозможность отказа от переданного и подписанного сообщения.

Справедливости ради следует заметить, что первые результаты по криптографии с открытым ключом были ранее получены британскими учеными, но они не были опубликованы [4].

Криптосистема RSA

Алгоритм *RSA* является первым алгоритмом шифрования с открытым ключом. Название системы *RSA* происходит от первых букв фамилий

ее авторов – Р. Ривест, А. Шамир и Л. Адлеман. Система базируется на следующих фактах:

- при известных числах d и b вычисление числа a из сравнения

$$a \equiv b^d \bmod n$$

по составному модулю n – это простая задача;

- вычисление неизвестного числа b при известных числах d и a из сравнения по составному модулю n

$$a \equiv b^d \bmod n$$

является трудной задачей;

- если известно, что p и q простые числа и $n = pq$, то вычислить n легко, а найти разложение n на простые множители трудно;
- если известно разложение $n = pq$ на простые множители, то задача вычисления числа b из уравнения

$$A \equiv b^d \bmod n$$

выполнима.

Теоретической основой криптосистемы *RSA* является теорема Эйлера из теории чисел. Напомним ее.

Теорема (Эйлера). Для любых натуральных и взаимно простых чисел n и a справедливо равенство

$$a^{\varphi(n)} \equiv 1 \bmod n.$$

Здесь $\varphi(n)$ есть функция Эйлера – количество взаимно простых с n натуральных чисел от 1 до n.

Из теории чисел известно, что если p и q простые числа, а $n = pq$, то

$$\varphi(n) = (p - 1)(q - 1).$$

Кроме того, из теоремы Эйлера следует, что если некоторое число e взаимно просто с $\varphi(n)$, то уравнение

$$de \equiv 1 \bmod \varphi(n),$$

или иначе

$$de = k\,\varphi(n) + 1,$$

однозначно разрешается относительно d. Решение легко определяется расширенным алгоритмом Евклида.

Итак, если известно, что

$$de \equiv 1 \bmod \varphi(n),$$

а x – передаваемая информация, то справедливо равенство

$$(x^e)^d \bmod n \equiv (x^{k\varphi(n)+1}) \bmod n \equiv$$
$$\equiv (x^{\varphi(n)})^k x \bmod n \equiv x.$$

Фактически последнее соотношение является основой для формулировки системы *RSA*.

Формирование системы RSA

1. Выбираем два различных простых числа p и q.
2. Вычисляем $n = pq$ и

$$\varphi(n) = (p - 1)(q - 1).$$

3. Выбираем число e, взаимно простое с $\varphi(n)$.
4. Вычисляем число d из уравнения

$$de \equiv 1 \ mod \ \varphi(n).$$

5. Определяем открытые ключи e и n.
6. Определяем закрытые ключи d, p, q и $\varphi(n)$.

Алгоритм шифрования

1. Дан текст сообщения M.
Шифротекст C вычисляется по формуле

$$C = E_k(M) = M^e \bmod n.$$

Алгоритм дешифрования

1. Дан шифротекст C.
2. Текст сообщения M вычисляется по формуле

$$M = D_k(C) = C^d \bmod n = (M^e)^d \bmod n = M.$$

Цифровая подпись

1. Есть абонент A и текст для подписи M.
2. Определяются закрытые ключи системы *RSA*, а именно d, p, q и $\varphi(n)$.

3. Определяются открытые ключи e и n.

4. Закрытым ключом вычисляется

$$C = M^d \bmod n.$$

Сообщение C рассматривается как подпись абонента A, потому что закрытый ключ d известен только ему.

5. Проверка подписанного документа вычисляется по формуле

$$C^e = (M^d)^e \bmod n = M,$$

используя открытый ключ e.

Замечание. Схема подписи усложняется, если абоненты A и B имеют систему *RSA* с эксклюзивными модулями n_A и n_B соответственно. Если абонент A желает свое подписанное сообщение

$$C = M^d \bmod n_A$$

зашифровать открытым ключом абонента B, т. е. вычислить

$$C_1 = C^{e\beta} \bmod n_B,$$

чтобы M было доступно только абоненту B, то значение

$$C1^{d_B} \bmod n_B$$

не всегда будет равно сообщению

$$C = M^d \bmod n_A.$$

Для этого необходимо выполнение неравенства $n_A < n_B$. Чтобы в подобном режиме пересылки информации разрешить представленную коллизию, пользователям надо договориться о некотором пороге T и создавать два набора параметров – один с модулем меньшим T, другой с модулем большим T. В этом случае отправитель использует свой меньший модуль для подписи, а больший модуль получателя – для шифрования [5].

Пример [6]. Зашифруем аббревиатуру *RSA*. Пусть $p = 17$ и $q = 31$. Тогда $n = pq = 527$, причем имеет место

$$\varphi(n) = (p - 1)(q - 1) = 480.$$

Выберем число $e = 7$, взаимно простое с $\varphi(n)$. Решая уравнение

$$de \equiv 1 \bmod \varphi(n)$$

относительно d с помощью алгоритма Евклида, найдем число $d = 343$. Поскольку $-137 = 343 \bmod 480$, то выполняется равенство $d = 343$. Проверка:

$$7 \cdot 343 = 2401 \equiv 1 \bmod 480.$$

Представим сообщение *RSA* в виде последовательности чисел, содержащихся в интервале [0, 526]. Для этого буквы *R*, *S* и *A* закодируем пятимерными двоичными векторами, воспользовавшись двоичной записью их порядковых номеров в английском алфавите

$$R = 18 = (10010),\ S = 19 = (10011),$$

$$A = 1 = (00001).$$

Тогда

$$RSA = (100101001100001).$$

Укладываясь в заданный интервал [0, 526], получаем представление кода

$$RSA = (100101001),\ (100001) = (M_1 = 297,\ M_2 = 33).$$

Далее последовательно шифруем M_1 и M_2

$$C_1 = E_к(M_1) = M_1^{\,e} \equiv 2977 \bmod 527 = 474.$$

При этом мы воспользовались тем, что

$$297\ 7 = ((297\ 2)3\ 297) \bmod 527 =$$
$$= (200\ 3\ (\text{mod } 527)\ 297) \bmod 527,$$

$$C_2 = E_к\,(M_2) = M_2^{\,e} \equiv 33^7 \bmod 527 = 407.$$

В итоге получаем шифртекст $y_1 = 474$ и $y_2 = 407$.

При расшифровывании нужно выполнить определенную последовательность действий. Вычислить

$$D_k\,(C_1)^{343} = (C_1)^{343} \bmod 527.$$

Отметим, что при возведении в степень удобно воспользоваться тем, что

$$343 = 256 + 64 + 16 + 4 + 2 + 1.$$

На основании этого представления получаем

$$474\ 2 \equiv 174 \bmod 527,\ 474\ 4 \bmod 527 \equiv 237,$$

$$474\ 8 \bmod 527 \equiv 307,\ 474\ 16 \bmod 527 \equiv 443,$$

$$474\ 32 \bmod 527 \equiv 205,\ 474\ 64 \bmod 527 \equiv 392,$$

$$474\ 128 \bmod 527 \equiv 307,\ 474\ 256 \bmod 527 \equiv 443,$$

в силу чего имеет место

$$474^{343} \bmod 527 \equiv$$
$$\equiv (443 \cdot 392 \cdot 443 \cdot 237 \cdot 174 \cdot 474) \bmod 527 \equiv 297.$$

Аналогично

$$407^{343} \bmod 527 \equiv 33.$$

Возвращаясь к буквенной записи, получаем после расшифровывания *RSA*.

Следуя [6], проанализируем вопрос о стойкости системы *RSA*. Можно показать, что сложность нахождения секретного ключа системы *RSA* определяется сложностью разложения числа n на простые множители. В связи с этим нужно выбирать числа p и q таким образом, чтобы задача разложения числа n была достаточно сложна в вычислительном плане. Для этого рекомендуются выполнять следующие требования:

1) числа p и q должны быть достаточно большими, не слишком сильно отличаться друг от друга и в то же время быть не очень близкими друг другу;

2) числа p и q должны быть такими, чтобы наибольший общий делитель чисел $(p-1)$ и $(q-1)$ был небольшим; желательно, чтобы наибольший общий делитель имел вид $(p-1, q-1) = 2$;

3) числа p и q должны быть сильно простыми числами.

Определение [7]. Простое число p называется **сильно простым**, если выполняются условия:

$$p \equiv 1 \bmod r,$$

$$p \equiv s - 1 \bmod s,$$

$$r \equiv 1 \bmod t,$$

где p, r, s, t – большие простые числа.

Когда не выполнено хотя бы одно из указанных условий, имеются эффективные алгоритмы разложения n на простые множители (см., например [8 – 9]).

В настоящее время самые большие простые числа вида $n = p \cdot q$, которые удается разложить на множители известными методами, содержат в своей записи 140 десятичных знака. Поэтому согласно указанным рекомендациям числа p и q в системе *RSA* должны содержать не менее 100 десятичных знаков.

Следует подчеркнуть необходимость соблюдения осторожности в выборе модуля *RSA* (числа n) для каждого из корреспондентов сети. Читатель может самостоятельно убедиться в том, что, зная одну из трех величин p, q или $\varphi(n)$, можно легко найти секретный ключ *RSA*. Известно также, что, зная секретную экспоненту расшифровывания d, можно легко разложить модуль n на множители. В этом случае удается построить вероятностный алгоритм разложения n. Отсюда следует, что каждый корреспондент сети, в которой для шифрования используется система *RSA*, должен иметь свой уникальный модуль.

В самом деле, если в сети используется единый для всех модуль n, то такая организация связи не обеспечивает конфиденциальности, несмотря на то, что базовая система *RSA* может быть стойкой. Другими словами, говорят о несостоятельности протокола с общим модулем. Несостоятельность следует из того, что знание произвольной пары экспонент (e_i, d_i) позволяет, как было отмечено, разложить n на множители. Поэтому любой корреспондент данной сети имеет возможность найти секретный ключ любого другого корреспондента. Более того, это можно сделать даже без разложения n на множители (см., например [8]).

Как отмечалось ранее, системы шифрования с открытыми ключами работают сравнительно медленно. Для повышения скорости шифрования *RSA* на практике используют малую экспоненту зашифровывания e [6].

СТЕПЕННЫЕ ВЫЧЕТЫ

Пусть дан модуль n и некоторое число a, взаимно простое с модулем n. Рассмотрим последовательность степеней

$$a, a^2, \ldots, a^t, \ldots$$

Найдем наименьшее число k, при котором

$$a^k \equiv 1 \bmod n.$$

Определение. Показателем a по модулю n называется наименьшее положительное число k, для которого выполняется сравнение $a^k \equiv 1 \ mod \ n$.

В литературе это наименьшее число k иногда называется показателем, которому a принадлежит по модулю n, и число k обозначается следующим образом:

$$k = P_n(a).$$

Если модуль n фиксирован, то для краткости $P_n(a)$ обозначается через $P(a)$.

Пример. Найти показатель, которому принадлежит 3 по модулю 11, т. е. $P_{11}(3)$. Для этого вычисляем последовательность степеней по модулю 11, т. е.

$$3 \bmod 11 \equiv 3 \bmod 11,$$

$$3^2 \bmod 11 \equiv 9 \bmod 11,$$

$$3^3 \bmod 11 \equiv 27 \bmod 11 \equiv$$
$$\equiv 5 \bmod 11,$$

$$3^4 \bmod 11 \equiv (3^3 \bmod 11)\,(3 \bmod 11) \equiv$$
$$\equiv (5 \cdot 3) \bmod 11 \equiv 4 \bmod 11,$$

$$3^5 \bmod 11 \equiv (3^4 \bmod 11)\,(3 \bmod 11) \equiv$$
$$\equiv 4 \cdot 3 \bmod 11 \equiv 1 \bmod 11.$$

Ответ: $P_{11}(3) = 5$.

Пример. Найти показатель, которому принадлежит 2 по модулю 15, т. е. $P_{15}(2)$. Для этого вычисляем последовательность степеней по модулю 15, т. е.

$$2 \bmod 15 \equiv 2 \bmod 15,$$

$$2^2 \bmod 15 \equiv 4 \bmod 15 \equiv 4 \bmod 15,$$

$$2^3 \bmod 15 \equiv 8 \bmod 15 \equiv 8 \bmod 15,$$

$$2^4 \bmod 15 \equiv 16 \bmod 15 \equiv 1 \bmod 15.$$

Ответ: $P_{15}(2) = 4$.

Теорема. Если

$$a \bmod n \equiv b \bmod n,$$

то $P_n(a) = P_n(b)$.

Теорема. Если $k = P_n(a)$ – показатель, которому принадлежит a по модулю n и

$$a^t \bmod n \equiv 1 \bmod n,$$

то k делит t, т. е. $k|t$.

Теорема. Если $k = P_n(a)$ – показатель, которому принадлежит a по модулю n и $\varphi(n)$ – функция Эйлера, то k делит $\varphi(n)$, т. е. $k|\ \varphi(n)$.

Теорема. Пусть $k = P_n(a)$ – показатель, которому принадлежит a по модулю n. Сравнение

$$a^t \bmod n \equiv a^s \bmod n$$

справедливо тогда и только тогда, когда

$$t \bmod k \equiv s \bmod k.$$

Пусть дан модуль n, некоторое число a, взаимно простое с модулем n, и k – показатель, которому принадлежит a по модулю n. Рассмотрим последовательность степеней

$$a, a^2, \ldots, a^t, \ldots$$

Тогда справедлива приведенная ниже теорема.

Теорема. В последовательности степеней числа a все степени принадлежат k – классам, представителями (вычетами) которых являются числа

$$a, a^2, \ldots, a^k.$$

Таким образом, все степени числа a принадлежат k-классам, где k-показатель, которому принадлежит a по модулю n.

Пример. Рассмотрим модуль 21 и число $a = 2$. Определим показатель, которому принадлежит 2 по модулю 21. Имеем

$$2 \bmod 21 \equiv 2 \bmod 21,$$

$$2^2 \bmod 21 \equiv 4 \bmod 21,$$

$$2^3 \bmod 21 \equiv 8 \bmod 21 \equiv 8 \bmod 21,$$

$$2^4 \bmod 21 \equiv 16 \bmod 21 \equiv 16 \bmod 21,$$

$$2^5 \bmod 21 \equiv 32 \bmod 21 \equiv 11 \bmod 21,$$

$$2^6 \bmod 21 \equiv 64 \bmod 21 \equiv 1 \bmod 21.$$

Итак, искомый показатель k равен 6. Отсюда следует, что все числа вида 2^t принадлежат классам [1–2], [4], [8], [11], [16].

Теорема. Пусть k – показатель, которому принадлежит a по модулю n. Если числа s и k взаимно просты, т. е. $(s, k) = 1$, то k является показателем числа a^s по модулю n.

Пример. Рассмотрим модуль 19 и число $a = 5$. Определим показатель, которому принадлежит 5 по модулю 19. Имеем

$$5 \bmod 19 \equiv 5 \bmod 19,$$

$$5^2 \bmod 19 \equiv 25 \bmod 19 \equiv 6 \bmod 19,$$

$$5^3 \bmod 19 \equiv 125 \bmod 19 \equiv 11 \bmod 19,$$

$$5^4 \bmod 19 \equiv (5^3 \bmod 19)\,(5 \bmod 19) \equiv$$
$$\equiv 55 \bmod 19 \equiv 17 \bmod 19,$$

$$5^5 \bmod 19 \equiv (5^4 \bmod 19)\,(5 \bmod 19) \equiv$$
$$\equiv 85 \bmod 19 \equiv 9 \bmod 19,$$

$$5^6 \bmod 19 \equiv (5^5 \bmod 19)\,(5 \bmod 19) \equiv$$
$$\equiv 45 \bmod 19 \equiv 7 \bmod 19,$$

$$5^7 \bmod 19 \equiv (5^6 \bmod 19)\,(5 \bmod 19) \equiv$$
$$\equiv 35 \bmod 19 \equiv 16 \bmod 19,$$

$$5^8 \bmod 19 \equiv (5^5 \bmod 19)\,(5^3 \bmod 19) \equiv$$
$$\equiv 99 \bmod 19 \equiv 4 \bmod 19,$$

$$5^9 \bmod 19 \equiv (5^8 \bmod 19)\,(5 \bmod 19) \equiv$$
$$\equiv 20 \bmod 19 \equiv 1 \bmod 19.$$

Итак, искомый показатель $k = 9$. Отсюда следует, что показатели 1, 2, 4, 5, 7, 8 взаимно просты c^k. Поэтому числа

$$5^2 \bmod 19 \equiv 6,$$

$$5^4 \bmod 19 \equiv 17,$$

$$5^5 \bmod 19 \equiv 9,$$

$$5^7 \bmod 19 \equiv 16,$$

$$5^8 \bmod 19 \equiv 4$$

имеют тот же показатель k по модулю 19.

Теорема. Пусть k – показатель, которому принадлежит a по простому модулю p. Тогда классы

$$[a], [a^2], \ldots, [a^k]$$

представляют собой все решения сравнения

$$x^k \equiv 1 \bmod p.$$

Пример. Пусть простой модуль равен 11. Тогда

$$\varphi(11) = 11 - 1 = 10 = 2 \cdot 5.$$

Любое число a может принадлежать показателю 1, 2, 5 или 10, так как показатель k должен делить $\varphi(11)$. Возьмем число 3. Найдем показатель k, которому принадлежит 3 по модулю 11. Имеем

$$3^1 \bmod 11 \equiv 3 \bmod 11 \neq 1,$$

$$3^2 \bmod 11 \equiv 9 \bmod 11 \neq 1,$$

$$3^5 \bmod 11 \equiv 243 \bmod 11 \equiv 1 \bmod 11.$$

Итак, $k = 5$. Отсюда следует, что решениями уравнения

$$x^5 \equiv 1 \bmod 11$$

являются числа из классов

$$[3], [3^2], [3^3], [3^4], [3^5].$$

Например, решением могут быть следующие числа:

$$3 \in [3], 9 \in [3^2], 5 \in [3^3], 4 \in [3^4], 12 \in [3^5].$$

Рассмотрим при заданном модуле n все классы, взаимно простые с n. Напомним, что эти классы порождаются числами от 0 до n–1. Заметим, что если модуль равен простому числу, то все классы взаимно просты с модулем. Пусть k – показатель, которому принадлежит число a по модулю n. Обозначим через $\psi(k)$ число классов, взаимно простых с n, для которых показатель по модулю n равен k.

Пример. Рассмотрим простой модуль $n = 11$, тогда значение функции Эйлера $\varphi(11) = 10$. Показателями k для любых чисел a по модулю 11 могут быть только делители значения функции Эйлера $\varphi(11) = 10$, т. е. числа 1, 2, 5, 10. Определим для чисел 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (т. е. чисел, которые являются представителями классов, порожденными числом 10) показатели по модулю 11. Поскольку модуль $n = 11$ есть простое число, то все классы взаимно просты с n. Для каждого числа от 1 до 10 вычислим показатель по модулю 11.

Вычисляем показатель k для числа 1, т. е.

$$1^1 \bmod 11 \equiv 1 \bmod 11.$$

Получаем $k = 1$.

Вычисляем показатель k для числа 2:

$$2^3 \bmod 11 \equiv 8 \bmod 11,$$

$$2^4 \bmod 11 \equiv 16 \bmod 11 \equiv 5 \bmod 11,$$

$$2^5 \bmod 11 \equiv 32 \bmod 11 \equiv 10 \bmod 11,$$

$$2^6 \bmod 11 \equiv (2^5 \bmod 11)\,(2 \bmod 11) \equiv$$
$$\equiv 20 \bmod 11 \equiv 9 \bmod 11,$$

$$2^7 \bmod 11 \equiv (2^6 \bmod 11)\,(2 \bmod 11) \equiv$$
$$\equiv 18 \bmod 11 \equiv 7 \bmod 11,$$

$$2^8 \bmod 11 \equiv (2^7 \bmod 11)\,(2 \bmod 11) \equiv$$
$$\equiv 14 \bmod 11 \equiv 3 \bmod 11,$$

$$2^9 \bmod 11 \equiv (2^8 \bmod 11)\,(2 \bmod 11) \equiv$$
$$\equiv 6 \bmod 11,$$

$$2^{10} \bmod 11 \equiv (2^9 \bmod 11)\,(2 \bmod 11) \equiv$$
$$\equiv 12 \bmod 11 \equiv 1 \bmod 11.$$

Получаем $k = 10$.

Вычисляем показатель k для числа 3:

$$3^2 \bmod 11 \equiv 9 \bmod 11,$$

$$3^3 \bmod 11 \equiv 27 \bmod 11 \equiv 5 \bmod 11,$$

$$3^4 \bmod 11 \equiv (3^3 \bmod 11)\,(3 \bmod 11) \equiv$$
$$\equiv 15 \bmod 11 \equiv 4 \bmod 11,$$

$$3^5 \bmod 11 \equiv (3^4 \bmod 11)\,(3 \bmod 11) \equiv$$
$$\equiv 12 \bmod 11 \equiv 1 \bmod 11.$$

Получаем $k = 5$.
Вычисляем показатель k для числа 4:

$$4^2 \bmod 11 \equiv 16 \bmod 11 \equiv 5 \bmod 11,$$

$$4^3 \bmod 11 \equiv 64 \bmod 11 \equiv 9 \bmod 11,$$

$$4^4 \bmod 11 \equiv (4^3 \bmod 11)\,(4 \bmod 11) \equiv$$
$$\equiv 36 \bmod 11 \equiv 3 \bmod 11,$$

$$4^5 \bmod 11 \equiv (4^4 \bmod 11)\,(4 \bmod 11) \equiv$$
$$\equiv 12 \bmod 11 \equiv 1 \bmod 11.$$

Получаем $k = 5$.
Вычисляем показатель k для числа 5:

$$5^2 \bmod 11 \equiv 25 \bmod 11 \equiv 3 \bmod 11,$$

$$5^3 \bmod 11 \equiv (5^2 \bmod 11)\,(5 \bmod 11) \equiv$$
$$\equiv 15 \bmod 11 \equiv 4 \bmod 11,$$

$$5^4 \bmod 11 \equiv (5^3 \bmod 11)\,(5 \bmod 11) \equiv$$
$$\equiv 20 \bmod 11 \equiv 9 \bmod 11,$$

$$5^5 \bmod 11 \equiv (5^4 \bmod 11)\,(5 \bmod 11) \equiv$$
$$\equiv 45 \bmod 11 \equiv 1 \bmod 11.$$

Получаем $k = 5$.
Вычисляем показатель k для числа 6:

$$6^2 \bmod 11 \equiv 36 \bmod 11 \equiv 3 \bmod 11,$$

$$6^3 \bmod 11 \equiv (6^2 \bmod 11)(6 \bmod 11) \equiv$$
$$\equiv 18 \bmod 11 \equiv 7 \bmod 11,$$

$$6^4 \bmod 11 \equiv (6^3 \bmod 11)(6 \bmod 11) \equiv$$
$$\equiv 42 \bmod 11 \equiv 9 \bmod 11,$$

$$6^5 \bmod 11 \equiv (6^4 \bmod 11)(6 \bmod 11) \equiv$$
$$\equiv 54 \bmod 11 \equiv 10 \bmod 11,$$

$$6^6 \bmod 11 \equiv (6^5 \bmod 11)\ (6 \bmod 11) \equiv$$
$$\equiv 60 \bmod 11 \equiv 5 \bmod 11,$$

$$6^7 \bmod 11 \equiv (6^6 \bmod 11)\ (6 \bmod 11) \equiv$$
$$\equiv 30 \bmod 11 \equiv 8 \bmod 11,$$

$$6^8 \bmod 11 \equiv (6^7 \bmod 11)\ (6 \bmod 11) \equiv$$
$$\equiv 48 \bmod 11 \equiv 4 \bmod 11,$$

$$6^9 \bmod 11 \equiv (6^8 \bmod 11)\ (6 \bmod 11) \equiv$$
$$\equiv 24 \bmod 11 \equiv 2 \bmod 11,$$

$$6^{10} \bmod 11 \equiv (6^9 \bmod 11)\ (6 \bmod 11) \equiv$$
$$\equiv 12 \bmod 11 \equiv 1 \bmod 11.$$

Получаем $k = 10$.
Вычисляем показатель k для числа 7:

$$7^2 \bmod 11 \equiv 49 \bmod 11 \equiv 5 \bmod 11,$$

$$7^3 \bmod 11 \equiv (7^2 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 35 \bmod 11 \equiv 2 \bmod 11,$$

$$7^4 \bmod 11 \equiv (7^3 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 14 \bmod 11 \equiv 3 \bmod 11,$$

$$7^5 \bmod 11 \equiv (7^4 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 21 \bmod 11 \equiv 10 \bmod 11,$$

$$7^6 \bmod 11 \equiv (7^5 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 70 \bmod 11 \equiv 4 \bmod 11,$$

$$7^7 \bmod 11 \equiv (7^6 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 28 \bmod 11 \equiv 6 \bmod 11,$$

$$7^8 \bmod 11 \equiv (7^7 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 42 \bmod 11 \equiv 9 \bmod 11,$$

$$7^9 \bmod 11 \equiv (7^8 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 63 \bmod 11 \equiv 8 \bmod 11,$$

$$7^{10} \bmod 11 \equiv (7^9 \bmod 11)(7 \bmod 11) \equiv$$
$$\equiv 56 \bmod 11 \equiv 1 \bmod 11,$$

Получаем $k = 10$.
Вычисляем показатель k для числа 8:

$$8^2 \bmod 11 \equiv 64 \bmod 11 \equiv 9 \bmod 11,$$

$$8^3 \bmod 11 \equiv (8^2 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 72 \bmod 11 \equiv 6 \bmod 11,$$

$$8^4 \bmod 11 \equiv (8^3 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 48 \bmod 11 \equiv 4 \bmod 11,$$

$$8^5 \bmod 11 \equiv (8^4 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 32 \bmod 11 \equiv 10 \bmod 11,$$

$$8^6 \bmod 11 \equiv (8^5 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 80 \bmod 11 \equiv 3 \bmod 11,$$

$$8^7 \bmod 11 \equiv (8^6 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 24 \bmod 11 \equiv 2 \bmod 11,$$

$$8^8 \bmod 11 \equiv (8^7 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 16 \bmod 11 \equiv 5 \bmod 11,$$

$$8^9 \bmod 11 \equiv (8^8 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 40 \bmod 11 \equiv 7 \bmod 11,$$

$$8^{10} \bmod 11 \equiv (8^9 \bmod 11)(8 \bmod 11) \equiv$$
$$\equiv 56 \bmod 11 \equiv 1 \bmod 11,$$

Получаем $k = 10$.
Вычисляем показатель k для числа 9:

$$9^2 \bmod 11 \equiv 81 \bmod 11 \equiv 4 \bmod 11,$$

$$9^3 \bmod 11 \equiv (9^2 \bmod 11)(9 \bmod 11) \equiv$$
$$\equiv 36 \bmod 11 \equiv 3 \bmod 11,$$

$$9^4 \bmod 11 \equiv (9^3 \bmod 11)(9 \bmod 11) \equiv$$
$$\equiv 27 \bmod 11 \equiv 5 \bmod 11,$$

$$9^5 \bmod 11 \equiv (9^4 \bmod 11)(9 \bmod 11) \equiv$$
$$\equiv 45 \bmod 11 \equiv 1 \bmod 11,$$

Получаем $k = 5$.

Вычисляем показатель k для числа 10:

$$10^2 \bmod 11 \equiv 100 \bmod 11 \equiv 1 \bmod 11,$$

Получаем $k = 2$.

Итак, для модуля $n = 11$ имеются:

1 класс [1], показатель которого равен 1, т. е. $\psi(1) = 1$;

1 класс [10], показатель которого равен 2, т. е. $\psi(2) = 1$;

4 класса [3], [4], [5], [9], показатели которых равны 5, т. е. $\psi(5) = 4$;

4 класса [2], [6], [7], [8], показатели которых равны 10, т. е. $\psi(10) = 4$.

Рассмотрим теперь случай, когда число классов взаимно простых с модулем меньше, чем сам модуль.

Пример. Рассмотрим простой модуль $n = 20$, тогда значение функции Эйлера

$$\varphi(20) = \varphi(22 \times 5) = 22 - 1 \times (2 - 1) \times (5 - 1) = 8.$$

Перечислим классы, порожденные модулем $n = 20$, которые взаимно простые с модулем 20:

$$[1], [3], [7], [9], [11], [13], [17], [19].$$

Вычисляем показатель k для числа 1:

$$1^1 \bmod 20 \equiv 1 \bmod 20.$$

Получаем $k = 1$.

Вычисляем показатель k для числа 3:

$$3^2 \bmod 20 \equiv 9 \bmod 20 \equiv 9 \bmod 20,$$

$$3^3 \bmod 20 \equiv 27 \bmod 20 \equiv 7 \bmod 20,$$

$$3^4 \bmod 11 \equiv 81 \bmod 20 \equiv 1 \bmod 20.$$

Получаем $k = 4$.

Вычисляем показатель k для числа 7:

$$7^2 \bmod 20 \equiv 49 \bmod 20 \equiv 9 \bmod 20,$$

$$7^3 \bmod 20 \equiv (7^2 \bmod 20)(\ 7 \bmod 20) \equiv$$
$$\equiv 63 \bmod 20 \equiv 3 \bmod 20,$$

$$7^4 \bmod 11 \equiv (7^3 \bmod 20)(7 \bmod 20) \equiv$$
$$\equiv 21 \bmod 20 \equiv 1 \bmod 20.$$

Получаем $k = 4$.

Вычисляем показатель k для числа 9:

$$9^2 \bmod 20 \equiv 81 \bmod 20 \equiv 1 \bmod 20,$$

Получаем $k = 2$.
Вычисляем показатель k для числа 11:

$$11^2 \bmod 20 \equiv 121 \bmod 20 \equiv 1 \bmod 20,$$

Получаем $k = 2$.
Вычисляем показатель k для числа 13:

$$13^2 \bmod 20 \equiv 169 \bmod 20 \equiv 9 \bmod 20,$$

$$13^3 \bmod 20 \equiv (13^2 \bmod 20)(13 \bmod 20) \equiv$$
$$\equiv 117 \bmod 20 \equiv 17 \bmod 20,$$

$$13^4 \bmod 11 \equiv (13^3 \bmod 20)(13 \bmod 20) \equiv$$
$$\equiv 221 \bmod 20 \equiv 1 \bmod 20.$$

Получаем $k = 4$.
Вычисляем показатель k для числа 17:

$$17^2 \bmod 20 \equiv 289 \bmod 20 \equiv 9 \bmod 20,$$

$$17^3 \bmod 20 \equiv (17^2 \bmod 20)(17 \bmod 20) \equiv$$
$$\equiv 153 \bmod 20 \equiv 13 \bmod 20,$$

$$17^4 \bmod 11 \equiv (17^3 \bmod 20)(17 \bmod 20) \equiv$$
$$\equiv 221 \bmod 20 \equiv 1 \bmod 20.$$

Получаем $k = 4$.
Вычисляем показатель k для числа 19:

$$19^2 \bmod 20 \equiv 381 \bmod 20 \equiv 1 \bmod 20,$$

Получаем $k = 2$.
Итак, для модуля $n = 20$ имеется:
1 класс [1], показатель которого равен 1, т. е. $\psi(1) = 1$;
3 класса [9], [11], [19], показатели которых равны 2, т. е. $\psi(2) = 3$;
4 класса [3], [7], [13], [17], показатели которых равны 4, т. е. $\psi(4) = 4$.
Заметим, что показатели, которым принадлежат числа a по модулю

n, являются делителями $\varphi(n)$. Поэтому показатели надо искать среди чисел k, которые делят $\varphi(n)$. Из предыдущих примеров видно, что может быть несколько классов, которые взаимно просты с модулем и которые имеют один и тот же показатель. Справедливо следующее утверждение.

Теорема. Пусть k_i, $i = 1, 2, \ldots t$ – делители значения функции Эйлера $\varphi(n)$ для модуля n, где t – число делителей $\varphi(n)$. Тогда справедливо равенство

$$\psi(k_1) + \psi(k_2) + \ldots + \psi(k_t) = \varphi(n),$$

где $\psi(k_i)$ – число классов, которые взаимно просты с модулем n и показатель которых равен k_i, $i = 1, 2, \ldots t$.

ПЕРВООБРАЗНЫЕ КОРНИ

Определение. Класс $[a]$, где $(a, n) = 1$, называется первообразным корнем по модулю n, если показатель числа a по модулю n равен $\varphi(n)$ – значению функции Эйлера для модуля n.

Известно, что любой показатель k числа a по модулю n делит $\varphi(n)$. Поэтому, чтобы убедиться, что число a является первообразным корнем по модулю n, надо проверить, что для любого числа k делителя $\varphi(n)$ $a^k \neq 1 \bmod n$.

Пример. Рассмотрим модуль $n = 54$. Тогда

$$\varphi(54) = \varphi(2 \times 27) = \varphi(2 \times 33) =$$
$$= (2 - 1) \times 33 - 1(3 - 1) = 18 = 2 \times 3 \times 3.$$

Делителями числа 18 являются числа 1, 2, 6, 9, 18. Рассмотрим число 5. Проверяем, будут ли числа 1, 2, 6, 9 показателями для числа 5 по модулю 54. Имеем

$$5^1 \bmod 54 \equiv 5 \bmod 54,$$

$$5^2 \bmod 54 \equiv 25 \bmod 54,$$

$$5^3 \bmod 54 \equiv 125 \bmod 54 \equiv 17 \bmod 54,$$

$$5^6 \bmod 54 \equiv (17 \times 17) \bmod 54 \equiv$$
$$\equiv 289 \bmod 54 \equiv 19 \bmod 54,$$

$$5^9 \bmod 54 \equiv (5^6 \bmod 54)(5^3 \bmod 54) \equiv$$
$$\equiv 17 \times 19 \equiv 323 \bmod 54 \equiv 53 \bmod 54,$$

$$5^9 \bmod 54 \equiv (5^9 \bmod 54)2 \equiv$$
$$\equiv 53 \times 53 \equiv 280\ 9 \bmod 54 \equiv 1 \bmod 54.$$

Из приведенных вычислений ясно, что показатель, которому принадлежит число 5 по модулю 54, равен 18, т. е. значению $\varphi(54)$. Отсюда следует, что класс [5] является первообразным корнем по модулю 54.

Заметим, что первообразных корней по модулю n может и не быть. Напомним, что для модуля $n = 20$, рассматривая все классы взаимно простые с модулем, мы получили такие результаты:

для класса [1] показатель равен 1;

для классов [9], [11], [19] показатель равен 2;

для классов [3], [7], [13], [17] показатель равен 4.

Нет ни одного класса взаимно простого с модулем, у которого показатель равен $\varphi(20) = 8$. Отсюда следует вывод, что для модуля 20 первообразный корень отсутствует.

Теорема. По любому простому модулю p существует $\varphi(p - 1)$ классов первообразных корней.

ИНДЕКСЫ

Определение. Пусть числа a и b взаимно просты с n, т. е. $(a, n) = 1$, $(b, n) = 1$. Число s называется индексом b по модулю n и основанию a, если

$$a^s \equiv b \bmod n.$$

При фиксированном модуле n для индекса s принята такая форма записи:

$$s = \operatorname{ind}_a b.$$

Для данной формы из определения индекса получается следующее тождество:

$$a^{\operatorname{ind}_a b} \equiv b \bmod n.$$

Заметим, что если s индекс числа b по основанию a, т. е. $s = \operatorname{ind}_a b$, то s является индексом для любого числа β из класса $[b]$ по любому основанию α из класса чисел $[a]$. Понятие индекса представляет собой аналогию понятия логарифма.

Пример. Пусть даны модуль $n = 13$ и основание $a = 2$. Тогда имеем

$$2^1 \equiv 2 \bmod 13, \text{ т. е. } \operatorname{ind}_2 2 = 1;$$

$$2^2 \equiv 4 \bmod 13, \text{ т. е. } \operatorname{ind}_2 4 = 2;$$

$$2^3 \equiv 8 \bmod 13, \text{ т. е. } \operatorname{ind}_2 8 = 3;$$

$$2^4 \equiv 16 \bmod 13 \equiv 3 \bmod 13, \text{ т. е. } \operatorname{ind}_2 3 = 4;$$

$$2^5 \equiv 32 \bmod 13 \equiv 6 \bmod 13, \text{ т. е. } \operatorname{ind}_2 6 = 5;$$

$$2^6 \equiv 64 \bmod 13 \equiv 12 \bmod 13, \text{ т. е. } \operatorname{ind}_2 12 = 6;$$

$$2^{13} \equiv (2^6 2^6 2) \bmod 13 \equiv$$

$$\equiv 288 \bmod 13 \equiv 2 \bmod 13, \text{ т. е. } \operatorname{ind}_2 2 = 13;$$

Обратим внимание на два соотношения:

$$\operatorname{ind}_2 2 = 1 \text{ и } \operatorname{ind}_2 2 = 13,$$

которые показывают, что одно число может иметь разные индексы.

Пример. Дано: модуль $n = 21$, основание $a = 5$, тогда имеем

$$5^2 \equiv 25 \bmod 21 \equiv 4 \bmod 21, \text{ т. е. } \mathrm{ind}_5\, 4 = 2;$$

$$5^4 \equiv 625 \bmod 21 \equiv 16 \bmod 21, \text{ т. е. } \mathrm{ind}_5\, 16 = 4;$$

$$5^6 \equiv 5^4 5^2 \bmod 21 \equiv 64 \bmod 21 \equiv$$
$$\equiv 1 \bmod 21, \text{ т. е. } \mathrm{ind}_5\, 1 = 6;$$

Обратим внимание, что для модуля 21 не существует индекса для числа 2 по основанию 5, т. е. значения $\mathrm{ind}_5\, 2$. Действительно, не существует числа s, для которого сравнение

$$5^s \equiv 2 \bmod 21$$

справедливо. На основании приведенного примера возникает вопрос – при каких условиях индекс числа b по основанию a существует для данного модуля n?

Для решения этого вопроса напомним, что если число $g = \varphi(n)$, где $\varphi(n)$ – значение функции Эйлера для модуля n, является наименьшим из всех чисел, при котором справедливо сравнение

$$a^g \equiv 1 \bmod n,$$

то g называется первообразным корнем. В этом случае справедливо следующее утверждение.

Теорема. Пусть число g есть любой первообразный корень по модулю n. Для каждого числа b (взаимно простого с модулем n) существуют индексы по основанию g, т. е. найдутся такие числа s, что выполняется сравнение

$$g^s \equiv b \bmod n.$$

Множество всех таких индексов s для данного фиксированного b совпадает с неотрицательными числами некоторого класса по модулю $\varphi(n)$.

Теорема. Пусть g – первообразный корень по модулю n и b – число взаимно простое с модулем n. Тогда сравнение

$$c \equiv b \bmod n$$

справедливо тогда и только тогда, когда

$$\mathrm{ind}_g\, c \equiv \mathrm{ind}_g\, b \bmod \varphi(n).$$

Теорема. Пусть g – первообразный корень по модулю n, a и b – числа, взаимно простые с модулем n, т. е.

$$(a, n) = 1, (b, n) = 1.$$

Тогда

$$\text{ind}_g (a, b) \equiv (\text{ind}_g a + \text{ind}_g b) \bmod \varphi(n).$$

Теорема. Пусть g – первообразный корень по модулю n и a – число взаимно простое с модулем n, т. е.

$$(a, n) = 1.$$

Тогда

$$\text{ind}_g a^k \equiv (k \times \text{ind}_g a) \bmod \varphi(n).$$

Определение. Если

$$a/b \equiv k \bmod n, n > 1 \text{ и } (b, n) = 1,$$

где число b – взаимно простое с n, то за $\text{ind}_g(a/b)$ принимается значение $\text{ind}_g k$.

Теорема. Пусть g – первообразный корень по модулю n и b – число взаимно простое с модулем n, $n > 1$, т. е.

$$(b, n) = 1.$$

Тогда

$$\text{ind}_g(a/b) \equiv (\text{ind}_g a - \text{ind}_g b) \bmod \varphi(n).$$

Рассмотрим случай, когда модуль простое число. В этом случае известно, что по любому простому p существуют $\varphi(p - 1)$ классов первообразных корней. Если взять за основание какое-либо g из любого класса первообразных корней, то для любого числа b, которое не делится на p, существует индексы. Будем рассматривать наименьший из всех возможных индексов числа b по основанию a для простого модуля p. При этих условиях для простых модулей p иногда вычисляют таблицу индексов для всех целых чисел из интервала $[1, p - 1]$.

Кроме того, известно, если число p, $p > 2$ – простое, то для составных модулей вида p^t и $2p^t$ существуют первообразные корни. Поэтому для чисел взаимно простых с модулями p^t и $2p^t$ существуют индексы.

Пример. Пусть дан модуль $n = 27 = 3^3 = p^3$, где p – простое число. Вычислим

$$\varphi(n) = \varphi(27) = \varphi(3^3) = 3^{3-1}(3-1) = 9 \times 2 = 18.$$

Известно, что показатели k, для которых

$$a^k \equiv 1 \bmod n$$

должны быть делителями $\varphi(n)$. Делителями числа 18 являются числа 1, 2, 3, 6, 9. Однако существует теорема, которая утверждает, что первообразные корни по модулю p^2 будут первообразными корнями и по модулю p^t, $t \geq 2$. По этой логике для того, чтобы найти показатели для некоторого числа a, для которого выполняется сравнение $a^k \equiv 1 \bmod n$ по модулю $n = 27$, достаточно рассмотреть только показатели, которые являются делителями значения функции Эйлера для числа $3^2 = 9$. Имеем

$$\varphi(9) = \varphi(3^2) = 3^{2-1} \times (3-1) = 3 \times 2 = 6.$$

Итак, среди показателей $k < \varphi(9)$ числа a, для которых выполняется сравнение

$$a^k \equiv 1 \bmod 9,$$

могут быть только делители числа $\varphi(9) = 6$. Отсюда следует, что для любого числа a показатель $k < \varphi(9)$, для которого выполняется сравнение $a^k \equiv 1 \bmod n$, могут быть только числа 1, 2, 3.

Рассмотрим число $g = 5$. Проверим, является ли число g первообразным корнем по модулю $n = 27$. Для этого вычислим g^1, g^2 и g^3:

$$5^1 \bmod 9 \equiv 5 \bmod 9,$$

$$5^2 \bmod 9 \equiv 25 \bmod 9 \equiv$$
$$\equiv (2 \times 9 + 7) \bmod 9 \equiv 7 \bmod 9,$$

$$5^3 \bmod 9 \equiv 125 \bmod 9 \equiv$$
$$\equiv (13 \times 9 + 8) \bmod 9 \equiv 8 \bmod 9.$$

По теории в условиях данной задачи существует такое k, для которого

$$5^k \bmod 9 \equiv 1 \bmod 9,$$

причем показатель k является делителем $\varphi(9)$. Из выкладок следует, что таким показателем может быть только число $\varphi(9) = 6$. Проверим данный вывод:

$$5^6 \bmod 9 \equiv (5^3 \times 5^3) \bmod 9 \equiv (8 \times 8) \bmod 9 \equiv$$
$$\equiv 64 \bmod 9 \equiv (7 \times 9 + 1) \bmod 9 \equiv 1 \bmod 9.$$

Из вычисления $5^6 \bmod 9$ следует, что 5 есть первообразный корень по модулю 9, а значит, число $g = 5$ является первообразным корнем и по модулю $n = 27$.

Убедившись, что $g = 5$ первообразный корень по модулю $n = 27$, т. е.

$$5^{\varphi(27)} \bmod 27 \equiv 5^{18} \bmod 27 \equiv 1 \bmod 27,$$

перейдем к вычислению индексов по модулю 27 с основанием $g = 5$. Имеем следующие индексы чисел по основанию $g = 5$ по модулю 27:

$$5^0 \bmod 27 \equiv 1 \bmod 27,$$

$$5^1 \bmod 27 \equiv 5 \bmod 27,$$

$$5^2 \bmod 27 \equiv 25 \bmod 27,$$

$$5^3 \bmod 27 \equiv 125 \bmod 27 \equiv$$
$$\equiv (4 \times 27 + 17) \bmod 27 \equiv 17 \bmod 27,$$

$$5^4 \bmod 27 \equiv 625 \bmod 27 \equiv$$
$$\equiv (23 \times 27 + 4) \bmod 27 \equiv 4 \bmod 27,$$

$$5^5 \bmod 27 \equiv (5^4 \times 5) \bmod 27 \equiv$$
$$\equiv (4 \times 5) \bmod 27 \equiv 20 \bmod 27,$$

$$5^6 \bmod 27 \equiv (5^5 \times 5) \bmod 27 \equiv$$
$$\equiv (20 \times 5) \bmod 27 \equiv 100 \bmod 27 \equiv 19 \bmod 27,$$

$$5^7 \bmod 27 \equiv (5^5 \times 5^2) \bmod 27 \equiv$$
$$\equiv (20 \times 25) \bmod 27 \equiv 500 \bmod 27 \equiv 14 \bmod 27,$$

$$5^8 \bmod 27 \equiv (5^4 \times 5^4) \bmod 27 \equiv$$
$$\equiv (4 \times 4) \bmod 27 \equiv 16 \bmod 27,$$

$$5^9 \bmod 27 \equiv (5^8 \times 5) \bmod 27 \equiv$$
$$\equiv (16 \times 5) \bmod 27 \equiv 80 \bmod 27 \equiv 26 \bmod 27,$$

$$5^{10} \bmod 27 \equiv (5^9 \times 5) \bmod 27 \equiv$$
$$\equiv (26 \times 5) \bmod 27 \equiv 130 \bmod 27 \equiv 22 \bmod 27,$$

$$5^{11} \bmod 27 \equiv (5^{10} \times 5) \bmod 27 \equiv$$
$$\equiv (22 \times 5) \bmod 27 \equiv 110 \bmod 27 \equiv 2 \bmod 27,$$

$$5^{12} \bmod 27 \equiv (5^{11} \times 5) \bmod 27 \equiv$$
$$\equiv (2 \times 5) \bmod 27 \equiv 10 \bmod 27,$$

$$5^{13} \bmod 27 \equiv (5^{12} \times 5) \bmod 27 \equiv$$
$$\equiv (10 \times 5) \bmod 27 \equiv 50 \bmod 27 \equiv 23 \bmod 27,$$

$$5^{14} \bmod 27 \equiv (5^{12} \times 5^2) \bmod 27 \equiv$$
$$\equiv (10 \times 25) \bmod 27 \equiv 250 \bmod 27 \equiv 7 \bmod 27,$$

$$5^{15} \bmod 27 \equiv (5^{14} \times 5) \bmod 27 \equiv$$
$$\equiv (7 \times 5) \bmod 27 \equiv 35 \bmod 27 \equiv 8 \bmod 27,$$

$$5^{16} \bmod 27 \equiv (5^{15} \times 5) \bmod 27 \equiv$$
$$\equiv (8 \times 5) \bmod 27 \equiv 40 \bmod 27 \equiv 13 \bmod 27,$$

$$5^{17} \bmod 27 \equiv (5^{16} \times 5) \bmod 27 \equiv$$
$$\equiv (13 \times 5) \bmod 27 \equiv 65 \bmod 27 \equiv 11 \bmod 27,$$

$$5^{18} \bmod 27 \equiv (5^{17} \times 5) \bmod 27 \equiv$$
$$\equiv (11 \times 5) \bmod 27 \equiv 55 \bmod 27 \equiv 1 \bmod 27.$$

В итоге проделанных вычислений получаем следующие значения дискретных логарифмов по модулю 27 при основании 5:

для числа 1 $\text{ind}_5\, 1 = 0$, для числа 2 $\text{ind}_5\, 2 = 11$,
для числа 4 $\text{ind}_5\, 4 = 4$, для числа 5 $\text{ind}_5\, 5 = 1$,
для числа 7 $\text{ind}_5\, 7 = 14$, для числа 8 $\text{ind}_5\, 8 = 15$,
для числа 10 $\text{ind}_5\, 10 = 12$, для числа 11 $\text{ind}_5\, 11 = 17$,
для числа 13 $\text{ind}_5\, 13 = 16$, для числа 14 $\text{ind}_5\, 14 = 7$,
для числа 16 $\text{ind}_5\, 16 = 8$, для числа 17 $\text{ind}_5\, 17 = 3$,
для числа 19 $\text{ind}_5\, 19 = 6$, для числа 20 $\text{ind}_5\, 20 = 5$,
для числа 22 $\text{ind}_5\, 22 = 10$, для числа 23 $\text{ind}_5\, 23 = 13$,
для числа 25 $\text{ind}_5\, 25 = 2$, для числа 26 $\text{ind}_5\, 26 = 9$.

В заключении рассмотрим сравнение

$$y \equiv g^x \bmod p.$$

Вычислить y при заданных значениях p, g и x не представляет труда. Обратная задача – по значениям p, g и y вычислить x, т. е. определить индекс или дискретный логарифм – трудная задача с точки зрения числа арифметических операций.

Сложность этой задачи для реально используемых значений модуля p находится за пределами возможностей современных вычислительных систем. Поэтому некоторые современные криптографические системы с открытыми ключами создаются на базе задачи дискретного логарифмирования. К подобным системам можно отнести схему Эль – Гамаля и схему Шнорра. Данные схемы используются для решения некоторых криптографических задач.

АЛГОРИТМ ДИСКРЕТНОГО ЛОГАРИФМИРОВАНИЯ

Задача дискретного логарифмирования, как и задача разложения на множители, применяется во многих алгоритмах криптографии с открытым ключом. Предложенная в 1976 году У. Диффи и М. Хеллманом для установления сеансового ключа, эта задача послужила основой для создания протоколов шифрования и цифровой подписи и других криптографических протоколов.

Задача дискретного логарифмирования в конечном поле

Определение 1. Задача дискретного логарифмирования формулируется так: для данных целых чисел a и b, $1 < a$, $b < p$ найти логарифм – такое целое число x, что

$$a^x \equiv b \pmod{p},$$

если такое число существует.

По аналогии с вещественными числами используется обозначение

$$x = \log_a b.$$

Безопасность соответствующих криптосистем основана на том, что, зная числа a, x, p, вычислить $a^x \pmod{p}$ легко (например, алгоритмом B-арного возведения в степень), а решить задачу дискретного логарифмирования трудно. Рассмотрим некоторые методы решения этой задачи.

ρ-метод Полларда

Рассмотренный в предыдущей главе ρ-метод Полларда может использоваться и для решения задачи дискретного логарифмирования. При этом случайное отображение f должно обладать не только сжимающими свойствами, но и вычислимостью логарифма (логарифм числа $f(c)$ можно выразить через неизвестный логарифм x и $\log_a f(c)$. Для дискретного логарифмирования в качестве случайного отображения f чаще всего используют «ветвящиеся» отображения, например

$$f(c) = ac \quad \text{если } c < p/2$$

или

$$f(c) = bc \quad \text{если } c > p/2.$$

При $c < p/2$ имеем $\log_a f(c) = \log_a c + 1$, при $c > p/2$ – $\log_a f(c) = \log_a c + x$.

Алгоритм 1. Дискретное логарифмирование ρ-методом Полларда.

Вход. Простое число p, число a порядка r по модулю p, целое число b, $1 < b < p$; отображение f, обладающее сжимающими свойствами и сохраняющее вычислимость логарифма.

Выход. Показатель x, для которого $a^x \equiv b \pmod p$, если такой показатель существует.

1. Выбрать произвольные целые числа u, v и положить $c \leftarrow a^u b^v \pmod p$, $d \leftarrow c$.

2. Выполнять $c \leftarrow f(c) \pmod p$, $d \leftarrow f(f(d)) \pmod p$, вычисляя при этом логарифмы для c и d как линейные функции от x по модулю r, до получения равенства $c \equiv d\ (p)$.

Приравняв логарифмы для c и d, вычислим логарифм x решением сравнения по модулю r. Результат: x или «решений нет».

Пример. Используя ρ-метод Полларда, решим задачу дискретного логарифмирования $10^x \equiv 64 \bmod 107$. Порядок числа 10 по модулю 107 равен 53.

Выберем отображение

$$f(c) \equiv 10c \bmod 107 \text{ при } c < 53,$$
$$f(c) \equiv 64 \bmod 107 \text{ при } c \geq 53. \text{ Пусть } u = 2,\ v = 2.$$

Результаты вычислений запишем в таблицу:

Номер шага	c	$\log_a c$	d	$\log_a d$
0	4	$2 + 2x$	4	$2 + 2x$
1	40	$3 + 2x$	79	$4 + 2x$
2	79	$4 + 2x$	56	$5 + 3x$
3	27	$4 + 3x$	75	$5 + 5x$
4	56	$5 + 3x$	3	$5 + 7x$
5	53	$5 + 4x$	86	$7 + 7x$
6	75	$5 + 5x$	42	$8 + 8x$
7	92	$5 + 6x$	23	$9 + 9x$
8	3	$5 + 7x$	53	$11 + 9x$
9	30	$6 + 7x$	92	$11 + 11x$
10	86	$7 + 7x$	30	$12 + 12x$
11	47	$7 + 8x$	47	$13 + 13x$

Приравниваем логарифмы, полученные на 11-м шаге:

$$7 + 8x \equiv 13 + 13x \bmod 53.$$

Решая сравнение первой степени, получаем $x \equiv 20 \bmod 53$.

Проверка: $10^{20} \equiv 64 \pmod{107}$.

Схема Эль – Гамаля

Алгоритм формирования схемы Эль – Гамаля

1. Выбираем простое число p.
2. Выбираем два случайных числа g, $g < p$ и x, $x < p$.
3. Вычисляем

$$y = g^x \bmod p.$$

Открытым ключом схемы Эль – Гамаля являются числа y, g и p, закрытым ключом – число x.

Алгоритм формирования цифровой подписи

1. Дано сообщение M, которое надо подписать.
2. Выбираем случайное число k, $k < p$, взаимно простое с $(p - 1)$.
3. Вычисляем

$$A = g^k \bmod p.$$

4. Из уравнения

$$M = (xa + kb) \bmod (p - 1)$$

определяем b по формуле

$$b = (M - xa)k^{-1} \bmod (p - 1).$$

5. Формируем подпись. Подписью будет пара чисел (a, b).

Число k является секретным ключом.

Замечание. При формировании цифровой подписи на четвертом шаге вместо значения M можно использовать значение

$$\mu = H(M),$$

где H – некоторая хэш-функция.

Проверка подписи

1. Дано сообщение M и подпись (a, b).
2. Вычисляем

$$y^a a^b \bmod p = \left[g^x\right]^a a^b = g^{xa} g^{kb} = g^{xa + kb} = g^M \bmod p.$$

3. Вычисляем $g^M \bmod p$.

Если значение

$$y^a a^b \bmod p$$

совпало с

$$g^M \bmod p,$$

то подпись верна.

Шифровка по алгоритму Эль – Гамаля

1. Дано сообщение M.
2. Выбираем случайное число k, $k < p$, взаимно простое с $(p - 1)$.
3. Вычисляем

$$a = g^k \bmod p.$$

4. Шифруем сообщение по формуле

$$b = y^k M \bmod p = g^{kx} M \bmod p.$$

Пара сообщений (a, b) является шифротекстом. Заметим, что шифротекст при таком способе шифрования длиннее исходного текста.

Расшифровка по алгоритму Эль – Гамаля

1. Вычисляем

$$ax \equiv g^{kx} \bmod p.$$

2. Тогда

$$M = b/a^x \equiv g^{kx} M / g^{kx} \bmod p.$$

Схема Шнорра

Алгоритм формирования схемы Шнорра

1. Выбираем два простых числа p и q таким образом, чтобы q было множителем $(p - 1)$.
2. Выбираем значение a такое, что $a \neq 1$ и

$$a^q \equiv 1 \bmod p.$$

3. Выбираем случайное число s, $s < q$.
4. Вычисляем

$$y = a^{-s} \bmod p.$$

Открытым ключом схемы Шнорра являются числа p, q и a, закрытым ключом – число y.

Алгоритм формирования цифровой подписи

1. Дано сообщение M и хеш-функция $H(M)$.
2. Выбираем случайное число r, $r < q$.
3. Вычисляем

$$x = a^r \bmod p.$$

4. Присоединяем к сообщению M значение x и вычисляем

$$E = H(M, x) \bmod q.$$

5. Вычисляем значение

$$V = (r + se) \bmod q.$$

6. Формируем подпись. Подписью является пара чисел (e, v).

Замечание. Число r – секретное число.

Проверка подписи

1. Дано сообщение M и подпись (e, v).
2. Вычисляется значение

$$x^1 = a^v y^e \bmod p,$$

где $a^v y^e = a^{(r + se) \bmod q} a^{-se \bmod q} = a^\gamma$.

3. Присоединяем к сообщению M значение x^1 и вычисляем

$$e^1 = H(M, x^1) \bmod q.$$

4. Если $e^1 = e$, то подпись считается верной.

Замечание. Числа r и s, которые используются в пункте 2,есть закрытые ключи. Они используются только для проверки выкладки.

Протокол аутентификации

1. Есть отправитель A и получатель B.
2. Пользователь A выбирает случайное число r, $r < q$ и вычисляет

$$x = a^r \bmod p.$$

3. Пользователь A отсылает B число x.
4. Пользователь B выбирает случайно число e из диапазона

$$[0, 2^t - 1].$$

5. Пользователь B отсылает A число e.
6. Пользователь A вычисляет

$$v = (r + se) \bmod q.$$

7. Пользователь A отсылает B число v.

8. Пользователь B вычисляет

$$x^1 = a^v y^e \bmod p,$$

где $a^v y^e = a^{(r+se) \bmod q} \; a^{-se \bmod q} = a^\gamma$.

9. Если $x_1 = x$, то доказательство аутентификации принимается.

Замечание 1. При протоколе аутентификации случайное число e – в отличии от цифровой подписи – определяет получатель B.

Замечание 2. От параметра t зависит безопасность алгоритма аутентификации. Шнорр [2; 4] предлагает использовать значение p порядка 512 битов, q – 140 битов, а значение числа t, равное 72.

КИТАЙСКАЯ ТЕОРЕМА ОБ ОСТАТКАХ

Одним из важных результатов теории чисел является так называемая китайская теорема об остатках (КТО). По существу эта теорема утверждает, что можно восстановить целое число по множеству его остатков от деления на числа из некоторого набора попарно взаимно простых чисел. Эта теорема была доказана приблизительно в 100 году до н. э. Есть несколько формулировок китайской теоремы об остатках. Представим здесь некоторые из них. Обозначим через Z_n кольцо вычетов по модулю n.

Теорема. Если $n = n_1 n_2 \ldots n_k$, где n_i, $1 \le I \le k$, – взаимно простые числа, то кольцо Z_n является прямой суммой колец Z_{ni}.

Теорема. Пусть n_i, $1 \le i \le k$, взаимно простые числа и пусть a_i целые числа. Тогда существует такое число x, что имеет место

$$x \equiv a_1 \bmod n_1,$$

$$x \equiv a_2 \bmod n_2,$$

$$\ldots$$

$$x \equiv a_k \bmod n_k.$$

Теорема. Пусть модули системы линейных сравнений являются взаимно простыми. Это означает, что общий делитель двух чисел n_i и n_j равен единице, т. е. $(n_i, n_j) = 1$ при $1 \le i, j \le k$. В этом случае существует класс вычетов Z_{ni}, который удовлетворяет условию

$$Z_{ni} \equiv 0 \bmod n_j.$$

Наконец, рассмотрим еще одну формулировку теоремы, которую будем использовать в практических работах.

Теорема. Пусть m_i, $1 \le I \le k$, – взаимно простые числа и

$$M = m_1 m_2 \ldots m_k$$

Пусть

$$a_i, 0 \le a_i \le m_i,$$

целые числа. Введем обозначение $M_i = M/m_i$. Пусть N_i число, которое удовлетворяет сравнению

$$M_i N_i \equiv 1 \ mod \ m_i.$$

При этих условиях сравнение

$$x \equiv a_i \bmod m_i$$

имеет на интервале [0, $M - 1$] единственное решение, которое определяется формулой

$$x = a_1N_1M_1 + a_2N_2M_2 + \ldots + a_kN_kM_k.$$

В рамках условий теоремы китайская теорема об остатках утверждает, что существует взаимно однозначное соответствие между целыми числами и некоторым наборами целых чисел. Другими словами, для каждого целого числа B найдется соответствующий ему единственный набор чисел

$$b_1, b_2, \ldots, b_k,$$

и наоборот, для каждого набора чисел ($b_1, b_2, \ldots, b_k$) найдется единственное соответствующему этому набору число B.

Пример. Решить систему сравнений

$$x \equiv a_1 \bmod 4,$$

$$x \equiv a_2 \bmod 5,$$

$$x \equiv a_3 \bmod 7,$$

где $m_1 = 4$, $m_2 = 5$, $m_3 = 7$. Определим число

$$M = m_1m_2m_3 = 4 \times 5 \times 7 = 140.$$

Вычислим

$$M_1 = M/m_1 = 140/4 = 35,$$

$$M_2 = M/m_2 = 140/5 = 28,$$

$$M_3 = M/m_3 = 140/7 = 20.$$

Вычислим N_1, N_2, и N_3 из следующих сравнений:

$$N_1M_1 = 35N_1 \equiv 1 \bmod 4,$$

$$N_2M_2 = 28N_2 \equiv 1 \bmod 5,$$

$$N_3M_3 = 20N_3 \equiv 1 \bmod 7.$$

Решаем первое сравнение

$$35N_1 \equiv 1 \bmod 4.$$

Определяем функцию Эйлера

$$\varphi(4) = \varphi(2^2) = 2^{2-1}(2 - 1) = 2.$$

Тогда из теоремы Ферма следует:

$$N_1 = 35^{\varphi(4)-1} \bmod 4 = 35^{2-1} \bmod 4 = 35 \bmod 4 = 3.$$

Проверка решения первого сравнения $35N_1 \equiv 1 \bmod 4$. Подставляем значение $N_1 = 3$ в сравнение

$$35N_1 \equiv 1 \bmod 4,$$

получаем

$$35 \times 3 \bmod 4 \equiv 105 \bmod 4 \equiv 1 \bmod 4.$$

Решаем второе сравнение

$$28N_2 \equiv 1 \bmod 5.$$

Определяем функцию Эйлера $\varphi(5) = 5 - 1 = 4$. Тогда из теоремы Ферма следует

$$N_2 = 28^{\varphi(5)-1} \bmod 5 = 28^{4-1} \bmod 4 =$$
$$=28^3 \bmod 5 = 112 \bmod 5 = 2 \bmod 5.$$

Проверка решения второго сравнения $28N_2 \equiv 1 \bmod 5$. Подставляем значение $N_2 = 2$ в сравнение

$$28N_2 \equiv 1 \bmod 5,$$

получаем

$$28 \times 2 \bmod 5 \equiv 56 \bmod 5 \equiv 1 \bmod 5.$$

Решаем третье сравнение

$$20N_3 \equiv 1 \bmod 7.$$

Определяем функцию Эйлера $\varphi(7) = 7 - 1 = 6$. Тогда из теоремы Ферма следует

$$N_3 = 20^{\varphi(7)-1} \bmod 7 = 20^{6-1} \bmod 7 =$$
$$= 20^5 \bmod 7 = 20 \bmod 7 = 6 \bmod 7.$$

Проверка решения третьего сравнения $20N_3 \equiv 1 \bmod 7$. Подставляем значение $N_3 = 6$ в сравнение

$$20N_2 \equiv 1 \bmod 7,$$

имеем

$$20 \times 6 \bmod 7 \equiv 120 \bmod 7 \equiv 1 \bmod 7.$$

Окончательно получаем, что решение системы сравнений

$$x \equiv a_1 \bmod 4,$$

$$x \equiv a_2 \bmod 5,$$

$$x \equiv a_3 \bmod 7,$$

определяется формулой

$$x = (a_1 N_1 M_1 + a_2 N_2 M_2 + a_3 N_3 M_3 =$$
$$= (35 \times 3a_1 + 28 \times 2a_2 + 20 \times 6a_3) \bmod 140.$$

СРАВНЕНИЯ СТЕПЕНЕЙ ВЫШЕ ПЕРВОГО

Переходя от сравнений первой степени к сравнениям более высоких степеней, целесообразно сначала рассмотреть тот случай, когда модуль – простое число. В этом случае имеется ряд весьма важных теорем, которые, вообще говоря, неверны для составных модулей. Вместе с тем теория сравнений по простому модулю является основой, на которой строится изучение сравнений по составному модулю. Во всей этой главе буквой p будем обозначать модуль, представляющий собой простое число.

Рассмотрим многочлен с целыми коэффициентами:

$$f(x) = c_0x^n + c_1x^{n-1} + \ldots + c_n.$$

В теории чисел изучают сравнения вида

$$f(x) \equiv g(x) \bmod m,$$

где $f(x)$, $g(x)$ – многочлены с целыми коэффициентами, а m – целое число. Сформулируем некоторые определения и результаты, связанные со сравнением $f(x) \equiv g(x) \bmod m$.

Решением сравнения $f(x) \equiv g(x) \bmod m$ называется класс по модулю m, который состоит из чисел, удовлетворяющих этому сравнению.

Если два сравнения подобного типа имеют одни и те же решения, то они называются эквивалентными.

Для сравнения $f(x) \equiv g(x) \bmod m$ справедливы те же утверждения, которые справедливы для сравнения вида $a \equiv b \bmod m$, где a, b, m – целые числа. Не нарушают эквивалентность сравнения следующие преобразования:

• к обеим частям сравнения можно прибавлять любой многочлен;

• обе части уравнения можно умножать на одно и то же целое число, которое взаимно простое с модулем;

• обе части уравнения и модуль можно умножать на одно и то же целое число $k > 0$.

Пусть даны два многочлена $f(x)$ и $g(x)$:

$$f(x) = a_0x^n + a_1x^{n-1} + \ldots + a_n,$$

$$g(x) = b_0x^n + b_1x^{n-1} + \ldots + b_n.$$

Справедливо следующее утверждение. Если $a_i \equiv b_i \bmod m$ ($i = 1, 2, \ldots, n$), то сравнения $f(x) \equiv 0 \bmod m$ и $g(x) \equiv 0 \bmod m$ – эквивалентны.

Теорема. Если $p \nmid c_0$, то сравнение

$$c_0x^n + c_1x^{n-1} + ... + c_n \equiv 0 \pmod{p}$$

может быть заменено эквивалентным сравнением с коэффициентом при старшем члене, равном единице.

Доказательство. Рассмотрим сравнение первой степени

$$c_0y \equiv 1 \pmod{p}.$$

Так как $p \nmid c_0$, то $(c_0\, p) = 1$ ($c_0\, p$ – взаимно просты), поэтому сравнение имеет решение. Найдем число y_0, удовлетворяющее этому сравнению, т. е. y_0 такое, что $c_0y_0 \equiv 1 \pmod{p}$. Тогда сравнение

$$c_0x^n + c_1x^{n-1} + \ldots + c_n \equiv 0 \pmod{p}$$

эквивалентно сравнению

$$(c_0\, y_0)\, x^n + (c_1\, y_0)\, x^{n-1} + ... + (c_n\, y_0) \equiv 0 \pmod{p},$$

а следовательно, и сравнению

$$x^n + b_1x^{n-1} + ... + b_n \equiv 0 \pmod{p},$$

где

$$b_1 \equiv c_1y_0 \pmod{p} \ldots b_n \equiv c_n\, y_0 \pmod{p}.$$

Пример. Заменить сравнение

$$27\, x^3 + 14\, x^2 - 10\, x + 13 = 0 \pmod{59}$$

эквивалентным сравнением с коэффициентом при старшем члене, равном 1. Решаем сравнение $27\, y_0 = 1 \pmod{59}$ и находим $y_0 - 35$. Данное нам сравнение эквивалентно сравнению

$$x^3 + 14 \cdot 35\, x^2 - 10 \cdot 35\, x + 13 \cdot 35 = 0 \pmod{59},$$

т. е. сравнению $x^3 + 18\, x^2 + 4\, x - 17 = 0 \pmod{59}$

Теорема. Если $f(x)$ и $g(x)$ многочлены с целыми коэффициентами, то сравнения по простому модулю:

$$f(x) \equiv 0 \pmod{p} \tag{1}$$

и

$$f(x) - (x^p - x_0)g(x) \equiv 0 \pmod{p}. \tag{2}$$

Доказательство. Пусть x_0 удовлетворяет сравнению (1), т. е. $f(x_0) \equiv 0(\mathrm{mod}\ p)$. Поскольку согласно теореме Ферма при любых x_0 имеет место $(x^p - x_0) \equiv 0(\mathrm{mod}\ p)$, то

$$f(x_0) - (x_0^p - x_0)g(x_0) \equiv 0(\mathrm{mod}\ p)\,.$$

Пользуясь той же теоремой Ферма, получаем, что если x_0 удовлетворяет сравнению (2), то $f(x_0) \equiv (x_0^p - x_0)g(x_0) \equiv 0(\mathrm{mod}\ p)$, и, таким образом, сравнения (1) и (2) эквивалентны. Из этой теоремы непосредственно вытекает следующая.

Теорема. Сравнение по простому модулю p, степень которого больше или равна, чем этот модуль, может быть заменено эквивалентным сравнением степени, меньшей, чем p.

Доказательство. Пусть $f(x)$ − многочлен с целыми коэффициентами степени $n \geq p$. При делении $f(x)$ на $x^p - x$, согласно известному способу деления многочлена на многочлен, неполное частное $g(x)$ и остаток $r(x)$ будут также многочлены с целыми коэффициентами

$$f(x) \equiv (x^p - x)g(x) + r(x)\,,$$

где степень $r(x)$ меньше степени $x^p - x$, т. е. меньше, чем p. Согласно предыдущей теореме сравнения $f(x) \equiv 0(\mathrm{mod}\ p)$ и $r(x) \equiv 0(\mathrm{mod}\ p)$ эквивалентны.

Замечание. Практически удобнее пользоваться теоремой 2, заменяя каждое слагаемое многочлена x^s, где $s \geq p$, слагаемым

$$x^s - (x^p - x)\, x^{s-p} = x^{s-(p-1)}$$

степени, меньшей, чем s. Следовательно, если $s = (p-1)\,g + r$ $(1 \leq r \leq p-1)$, то x^s можно заменить на x^r. Проделывая эту операцию для всех слагаемых многочлена, достигнем того же результата, что и применением теоремы 3.

Пример. Сравнение $x^{16} + 3x^8 - 5x^7 - x^4 + 6x - 2 \equiv 0\ (\mathrm{mod}\ 7)$ заменить эквивалентным сравнением степени, меньшей, чем 7.

Решение. Согласно только что сделанному замечанию мы получим эквивалентное сравнение, если заменим x^{16} на $x^{16-2\cdot 6}$, x^8 на x^2, x^7 на x. Таким образом, заданное сравнение эквивалентно сравнению

$$(x^4 + 3x^2 - 5x) - x^4 + 6x - 2 \equiv 0\ (\mathrm{mod}\ 7),$$

т. е. сравнению $3x^2 + x - 2 \equiv 0\ (\mathrm{mod}\ 7)$.

Теорема. Если $f(x)$, $g(x)$, $h(x)$, $r(x)$ − многочлены с целыми коэффициентами ($f(x) = g(x)h(x) + r(x)$) и все коэффициенты $r(x)$ делятся на простое число p, то любое решение сравнения

$$f(x) \equiv 0(\mathrm{mod}\ p) \qquad (3)$$

является решением, по крайней мере, одного из сравнений

$$g(x) \equiv 0(\mathrm{mod}\, p),\ \ h(x) \equiv 0(\mathrm{mod}\, p). \qquad (4)$$

Доказательство. Пусть x_0 – решение сравнения (3), т. е.

$$r(x_0) \equiv 0(\mathrm{mod}\, p).$$

Поскольку все коэффициенты $r(x)$ делятся на p, будем также иметь $r(x_0) \equiv 0 \pmod{p}$, а поэтому

$$g(x_0)h(x_0) = f(x_0)r(x_0) \equiv 0 \pmod{p}.$$

Из сравнимости произведения $g(x_0)h(x_0)$ с нулем по модулю p следует, что, по крайней мере, один из этих множителей сравним с нулем по данному модулю, т. е. x_0 − решение хотя бы одного из уравнений (4).

Пример. В сравнении $x^4 + 18x^2 + 5 \equiv 0 \pmod{31}$ левую часть можно представить в виде $(x^2 - 4x) - (x^2 + 9) + (31x^2 - 31)$. И мы находим все решения этого сравнения, решая сравнения $x^2 - 4 \equiv 0 \pmod{31}$, $x^2 - 9 \equiv 0 \pmod{31}$, т. е. $x \equiv \pm 2 \pmod{31}$ и $x \equiv \pm 3 \pmod{31}$. Все эти четыре класса удовлетворяют нашему сравнению.

Для составных модулей эта теорема неверна. Например: сравнению $x^2 + 4x = x\,(x + 4) \equiv 0 \pmod{12}$ удовлетворяет класс 6, не являющийся решением ни одного из сравнений: $x \equiv 0 \pmod{12}$, $x + 4 \equiv 0 \pmod{12}$.

Теорема. Сравнение степени n по простому модулю p с коэффициентом при старшем члене, не делящемся на p, может иметь не больше, чем n решений.

Доказательство. Утверждение теоремы верно при $n = 1$. Действительно, в этом случае мы имеем сравнение первой степени

$$c_0x + c_1 + \equiv 0 \pmod{p},$$

где $p \nmid c_0$, т. е. $(c_0, p) = 1$, а такое сравнение имеет одно решение. Применим теперь для доказательства теоремы метод полной математической индукции.

Предположим, что утверждение теоремы верно для всех многочленов $(n - 1)$-й степени со старшими коэффициентами, не делящимися на простой модуль p. Возьмем теперь произвольный многочлен n-й степени:

$$f(x) = c_0x^n + c_1x^{n-1} + \ldots + c_n,$$

где $p \nmid c_0$ и рассмотрим сравнение

$$f(x) \equiv 0 \pmod{p}. \qquad (5)$$

Если это сравнение не имеет ни одного решения, то число решений меньше, чем n. Если же это сравнение имеет решения, то возьмем любое число x_0, удовлетворяющее ему, и разделим $f(x)$ на $x - x_0$. Согласно теореме Безу будем иметь

$$f(x) = (x - x_0)g(x) + f(x_0).$$

Коэффициенты многочлена $(n - 1)$-й степени

$$g(x) = b_0x^{n-1} + \ldots + b_n$$

могут быть, как известно, найдены по схеме Горнера и представляют собой целые числа, причем $b_0 = c_0$. Поскольку x_0 удовлетворяет сравнению (5), $p|f(x_0)$, то все решения (5) находятся среди решений сравнений $x - x_0 \equiv 0 (\mathrm{mod}\ p)$ и $g(x) \equiv 0 (\mathrm{mod}\ p)$, удовлетворяя либо одному из них, либо обоим. Сравнение $x - x_0 \equiv 0 (\mathrm{mod}\ p)$ имеет одно решение, а сравнение $g(x) \equiv 0 (\mathrm{mod}\ p)$, представляющее собой сравнение $(n - 1)$-й степени по простому модулю с коэффициентом при старшем члене $b_0 = c_0$, не делящемся на p, согласно предположению, может иметь не больше, чем $n - 1$ решений. Таким образом, сравнение (5) имеет не больше, чем $1 + (n - 1)$, т. е. не больше, чем n, решений.

Утверждение теоремы было проверено при $n = 1$. Из справедливости утверждения для многочленов $(n - 1)$-й степени следует справедливость этого же утверждения для многочленов n-й степени. Согласно принципу полной математической индукции, справедливость теоремы доказана.

Пример. $x_0 = 31$ удовлетворяет сравнению $11x^2 \equiv 65 (\mathrm{mod}\ 103)$. Найти все решения этого сравнения.

Очевидно, что вместе с классом $\overline{31}$ этому сравнению удовлетворяет и класс $-\overline{31}$. Коэффициент при старшем члене 11 не делится на простой модуль 103, поэтому сравнение не может иметь больше двух решений.

Ответ. $x \equiv \pm 31 (\mathrm{mod}\ 103)$.

Для составных модулей эта теорема неверна. Сравнение степени n по составному модулю с коэффициентом при старшем члене, не делящемся на модуль или даже взаимно простом с модулем, может иметь больше, чем n, решений. Например, сравнение $x^2 - 3x + 2 = 0 (\mathrm{mod}\ 6)$ имеет 4 решения $\overline{1}, \overline{2}, \overline{4}, \overline{5}$.

Теорема 6. Если сравнение степени n по простому модулю p имеет больше, чем n, решений, то все коэффициенты сравнения делятся на p.

Доказательство. Возьмем любое простое число p. Если сравнение $c_0x + c_1 \equiv 0 (\mathrm{mod}\ p)$ имеет больше, чем одно решение, то $(c_0, p) \neq 1$. Таким образом, при $n = 1$ теорема верна. Предположим, что утверждение теоремы верно для многочленов степени, меньшей, чем n, т. е. предположим,

что число решений сравнения степени, меньшей, чем n, может превосходить степень сравнения только тогда, когда все коэффициенты делятся на модуль p. Возьмем любое сравнение степени n

$$c_0x^n + c_1x^{n-1} + ... + c_n \equiv 0 \pmod{p} \tag{6}$$

имеющее больше, чем n, решений. Согласно теореме 5 из этого раздела, в таком сравнении c_0 делится на p, а тогда сравнение

$$c_ix^{n-1} + ... + c_n \equiv 0 \pmod{p}, \tag{7}$$

эквивалентное сравнению (6), также имеет больше, чем n, решений.

В сравнении (7), степень которого меньше n, а число решений превосходит степень, согласно предположению, все коэффициенты должны делиться на p, т. е. $p|c_1$, ..., $p|c_n$. Поскольку уже раньше было установлено, что $p|c_0$, то утверждение теоремы верно для n. Согласно принципу полной математической индукции справедливость теоремы доказана.

Теорема. Пусть $f(x) = x^n + c_1x^{n-1} + ... + c_n$ – многочлен с целыми коэффициентами и свободным членом $c_n \neq 0 \pmod{p}$, где p – простое число, причем $p \geq n$. Сравнение $f(x) \equiv 0 \pmod{p}$ имеет n решений тогда и только тогда, когда все коэффициенты остатка от деления $x^{p-1} - 1$ на $f(x)$ кратны p.

Доказательство. Пусть

$$x^{p-1} - 1 = f(x)\, g(x) + r(x),$$

где $g(x)$ и $r(x)$ – многочлены с целыми коэффициентами, причем степень $r(x)$ меньше, чем n.

1. Докажем достаточность условия. Пусть коэффициенты $r(x)$ делятся на p. Обозначим через S и T соответственно число решений сравнений

$$f(x) \equiv 0 \pmod{p}, \tag{8}$$

$$g(x) \equiv 0 \pmod{p}. \tag{9}$$

Сравнение $x^{p-1} - 1 = 0 \pmod{p}$ по теореме Ферма имеет $p - 1$ решений. Каждое из этих $p - 1$ решений является решением хотя бы одного из сравнений: (8) или (9), т. е. $S + T \geq p - 1$. Сравнение (9) степени $(p - 1 - n)$ имеет коэффициент при старшем члене, равный единице, так что $T \leq p - 1 - n$ и, значит,

$$S \geq (p - 1) - T \geq p - 1 - (p - 1 - n) = n.$$

Поскольку при этом $S \leq n$, получаем $S = n$, т. е. из делимости коэффициентов $r(x)$ на p следует, что число решений сравнения (8) равно n.

2. Докажем необходимость условия. Пусть сравнение (8) имеет n решений. Если x_0 – решение сравнения (8), то $f(x_0) \equiv 0 \pmod p$, и вместе с тем, поскольку $p \neq c_n$, то $p \neq x_0$, значит, согласно теореме Ферма, $x_0^{p-1} - 1 = 0 \pmod p$, так что

$$r(x_0) = (x_0^{p-1} - 1) - f(x_0)g(x_0) = 0(\mathrm{mod}\, p).$$

Таким образом, каждое из n решений сравнения (8) является решением сравнения $r(x) \equiv 0 \pmod p$, степень которого меньше, чем n. Согласно теореме 6, все коэффициенты $r(x)$ делятся на p.

Пример: Сравнению $x^3 = 1 \pmod{13}$ удовлетворяют классы 1 и 3. Имеет ли это сравнение еще одно решение? Для $x^{12} - 1$ на $x^3 - 1$, находим

$$x^{12} - 1 = (x^3 - 1)(x^9 + x^6 + x + 1),$$

так что $r(x) = 0$ и, следовательно, это сравнение имеет три решения.

2. Некоторые из рассмотренных нами теорем можно легко обобщить на случай сравнений с несколькими неизвестными вида

$$f(x_1, x_2, ..., x_S) \equiv 0(\mathrm{mod}\, p), \qquad (10)$$

где $f(x_1, x_2, ..., x_S)$ – многочлен с целыми коэффициентами, а p – простое число. Непосредственным обобщением теоремы 3 является следующая.

Теорема. Если в левой части сравнения (10) некоторые из неизвестных встречаются в виде степени с показателем большем или равным p, то сравнение (10) можно заменить эквивалентным сравнением, в котором степень каждого из неизвестных не превосходит $p - 1$.

СРАВНЕНИЯ ПО СОСТАВНОМУ МОДУЛЮ

В этом разделе будут рассмотрены способы приведения сравнений по составному модулю к сравнениям по простому модулю. Следующая теорема показывает, что решение сравнений по модулю $m = p_1^{\alpha_1} \ldots p_s^{\alpha_s}$, где p_i – простые числа, может быть приведено к решению сравнений по модулям $p_i^{\alpha_i}$. Во всей этой главе $f(x)$ будет обозначать произвольный многочлен с целыми коэффициентами.

Теорема 1. Если. $m = p_1^{\alpha_1} \ldots p_s^{\alpha_s}$ – каноническое разложение модуля m, то сравнение

$$f(x) \equiv 0 \pmod{m} \tag{1}$$

эквивалентно системе сравнений

$$\left.\begin{array}{c} f(x) \equiv 0 (\mathrm{mod}\, p_1^{\alpha_1}) \\ \ldots\ldots \\ \ldots\ldots \\ f(x) \equiv 0 (\mathrm{mod}\, p_s^{\alpha_s}) \end{array}\right\}. \tag{2}$$

Доказательство. Решения системы (2) по определению есть классы по модулю m, равному наименьшему общему кратному чисел $p_1^{\alpha_1} \ldots p_s^{\alpha_s}$. Если класс a по модулю m удовлетворяет системе (2), т. е. если $p_1^{\alpha_1} \mid f(a) \ldots p_S^{\alpha_s} \mid f(a)$, то согласно теореме (32) $f(a)$ делится на m,

$$f(x) \equiv 0 \pmod{m},$$

т. е. а представляет собой решение сравнения (1). Наоборот, если класс $\overline{a}$ удовлетворяет сравнению (1), то $m \mid f(a)$, и поскольку $p_i^{\alpha_i} \mid m$, $p_i^{\alpha_i} \mid f(a)$, $f(a) \equiv 0 (\mathrm{mod}\, p_i^{\alpha_i})$ при $i = 1, 2, \ldots, s$, т. е. $\overline{a}$ – решение системы (2).

Для нахождения решений системы (2) обычно предварительно решают каждое из сравнений этой системы. Если хотя бы одно из сравнений (2) не имеет решений, то и вся система несовместна, т. е. в этом случае сравнение (1) не имеет решений. Если каждое из сравнений имеет хотя бы одно решение, то находим их в виде

$$\left.\begin{array}{l} f(x) \equiv a_1 (\mathrm{mod}\ p_1^{\alpha_1}) \\ \ldots\ldots \\ \ldots\ldots \\ f(x) \equiv a_s (\mathrm{mod}\ p_{1s}^{\alpha_s}) \end{array}\right\}. \qquad (3)$$

Значения x, удовлетворяющие всем этим сравнениям с взаимно простыми модулями, существуют и образуют класс по модулю m, что является решением (3), а следовательно, решением исходной системы (2) и сравнения (1).

Если некоторые из сравнений (2) имеют больше, чем по одному решению, то мы получим несколько систем вида (3). А именно, если сравнение $f(x) \equiv 0 (\mathrm{mod}\ p_1^{\alpha_1})$ имеет k_1 решений, $f(x) \equiv 0 (\mathrm{mod}\ p_2^{\alpha_2})$ имеет k_2 решений, ..., $f(x) \equiv 0 (\mathrm{mod}\ p_S^{\alpha_S})$ имеет k_s решений, то мы можем составить $k_1 \cdot k_2 \cdot \ldots \cdot k_s$ систем вида (3), каждая из которых даст по одному решению системы (2), и тогда система (2) и сравнение (1) имеют $k_1 \cdot k_2 \cdot \ldots \cdot k_s$ решений. Можно сформулировать полученный нами результат в виде следующей теоремы.

Теорема. Число решений сравнения (1) равно $k_1 \cdot k_2 \cdot \ldots \cdot k_s$ где $k_1 \cdot k_2 \cdot \ldots \cdot k_s$ соответственно равно числу решений каждого из сравнений (2).

Пример. Решить сравнение $x^2 - 3x + 23 \equiv 0 \pmod{63}$.

Сравнение эквивалентно системе:

$$x^2 - 3x + 23 \equiv 0 \pmod 7,$$

$$x^2 - 3x + 23 \equiv 0 \pmod 9.$$

Для сравнения $x^2 - 3x + 23 \equiv 0 \pmod 7$ находим два решения: $x \equiv 1 \pmod 7$ и $x \equiv 2 \pmod 7$, а для сравнения $x^2 - 3x + 23 \equiv 0 \pmod 9$ – два решения: $x \equiv 4 \pmod 9$ и $8 \pmod 9$. Решаем четыре системы:

1) $x \equiv 1 \pmod 7$, $x \equiv 4 \pmod 9$;
2) $x \equiv 1 \pmod 7$, $x \equiv 8 \pmod 9$;
3) $x \equiv 2 \pmod 7$, $x \equiv 4 \pmod 9$;
4) $x \equiv 2 \pmod 7$, $x \equiv 8 \pmod 9$.

и находим следующие решения: 1) $x \equiv 22 \pmod{63}$, 2) $x \equiv 8 \pmod{63}$, 3) $x \equiv 58 \pmod{63}$, 4) $x \equiv 44 \pmod{63}$.

ДВУЧЛЕННЫЕ СРАВНЕНИЯ

Определение 1. Двучленным сравнением называется сравнение вида

$$ax^n \equiv b \bmod m.$$

Мы рассмотрим двучленные сравнения по простому модулю $p > 2$ вида

$$x^n \equiv a \bmod p. \tag{1}$$

Теорема 1. Если $p \nmid a$, то сравнение (1) по простому модулю $p > 2$ либо совсем не имеет решений, либо число решений равно наибольшему общему делителю n и $p - 1$. Кроме того, если НОД$(n, p - 1) = d$, то сравнение не имеет решений, если d не делит ind a и имеет d решений, если d делит ind a.

Доказательство. Пусть $(n, p - 1) = d$. Тогда, как отмечалось выше, по простому модулю p существует первообразный корень. Индексируя сравнение (1) по некоторому основанию g, получаем

$$n \operatorname{ind} x = \operatorname{ind} a \bmod (p - 1). \tag{2}$$

Сравнения (1) и (2), согласно свойствам индексов, эквивалентны. Если обозначить ind $x = z$, ind $a = b$, то для неизвестной z получим сравнение первой степени

$$nz \equiv b \bmod (p - 1). \tag{3}$$

Если d не делит ind a (т. е. d не делит b), то сравнение (3) не имеет решений. Но тогда не существует и значений x, удовлетворяющих сравнениям (1) и (2).

При $d \nmid$ ind a (т. е. d не делит b) сравнение (3) имеет d решений. Значения ind x, удовлетворяющие сравнению (2), принадлежат d классам по модулю $(p - 1)$, а следовательно, и для x существует d классов по модулю p, удовлетворяющих сравнению (1).

Заметим, если p делит a, то сравнение $x^n \equiv a \bmod p$ эквивалентно сравнению $x^n \equiv 0 \bmod p$ и в этом случае оно имеет одно решение $x \equiv 0 \bmod p$.

Определение. Число a называется вычетом n-й степени по простому модулю p, если p не делит a и сравнение $x^n \equiv a \bmod p$ имеет решения.

Определение. Число a называется невычетом n-й степени по простому модулю p, если сравнение $x^n \equiv a \bmod p$ не имеет решений.

Заменяя в сравнении (1) a любым числом, сравнимым с ним по модулю p, получим сравнение, эквивалентное (1). Следовательно, в приведенных выше определениях вместо отдельных вычетов можно рассматривать соответствующие классы. При таком рассмотрении исключается класс чисел, равных 0.

Если a представляет собой вычет n-й степени по модулю p, то весь класс состоит из вычетов n-й степени по модулю p, а если a – невычет, то и весь соответствующий класс состоит из невычетов. Можно поэтому вместо отдельных вычетов и невычетов n-й степени рассматривать соответствующие классы.

Теорема. По простому модулю $p > 2$ число классов вычетов n-й степени равно $(p - 1)/d$, где

$$d = \text{НОД}(n, p - 1). \tag{4}$$

Теорема. Если $d = \text{НОД}(n, p - 1)$, то вычеты n-й степени по простому модулю $p > 2$ совпадают с вычетами степени d по этому модулю.

Теорема. При $p \nmid a$, $n \mid (p - 1)$ сравнение $x^n \equiv a \bmod p$ по простому модулю $p > 2$ либо совсем не имеет решений, либо имеет n решений. По такому модулю a является вычетом n-й степени тогда и только тогда, когда $n \mid \text{ind } a$.

Теорема. По простому модулю $p > 2$ и $n \mid (p - 1)$ число классов вычетов n-й степени равно $(p - 1)/n$.

В следующей теореме мы установим необходимое и достаточное условие того, чтобы a при $n \mid p - 1$ было вычетом n-й степени по модулю p.

Теорема. При условии $n \mid (p - 1)$ (n делит $p - 1$) число a является вычетом n-й степени по простому модулю $p > 2$ тогда и только тогда, когда

$$a^{(p-1)/n} \equiv 1 \bmod p. \tag{5}$$

Легко проверить, что теоремы справедливы также и при $p = 2$. Тогда $a \in \overline{1}$ и $p - 1 = 1$, так что утверждения этих теорем тривиальны.

Сравнение $x^n \equiv 1 \bmod p$, очевидно, имеет решение $x \equiv 1 \bmod p$. Согласно первой теореме, это сравнение будет, вообще говоря, иметь $d = (n, p - 1)$ решений, $d = \text{НОД}(n, p - 1)$. Если n делит $p - 1$, то сравнение будет иметь n решений. Естественно классы, удовлетворяющие этому сравнению, или, что то же самое, уравнению $\overline{x}^n = 1$, называть корнями n-й степени из единицы по модулю p. Аналогично этому классы (если они существуют), удовлетворяющие сравнению (1), т. е. уравнению $\overline{x}^n = \overline{a}$, естественно назвать корнями n-й степени из $\overline{a}$ по модулю p.

Определение. Корнем n-й степени из класса $\overline{a}$ по простому модулю p называется класс чисел по этому модулю, удовлетворяющий сравнению

$$x^n \equiv a \bmod p. \tag{6}$$

По простому модулю p, если $n \mid (p-1)$, либо не существует ни одного корня n-й степени, либо число корней равно n.

Теорема. При $n \mid p-1$, $p \nmid a$ все решения сравнения (1) по простому модулю $p > 2$ можно получить, умножая одно решение этого сравнения на различные решения сравнения $x^n \equiv 1 \bmod p$.

Другими словами, все корни n-й степени из $\overline{a}$ по модулю p можно получить, умножая один из этих корней на различные корни n-й степени из $\overline{1}$. Сравнение по простому модулю

$$ax^n \equiv b \bmod p$$

при условии $p \nmid a$ эквивалентно сравнению $x^n \equiv c \bmod p$, где $ac = b \bmod p$. При $p \nmid b$ будем иметь также $p \nmid c$, и следовательно, сравнение (7) либо совсем не имеет решений, либо число решений равно d, где $d = (n, p-1)$.

Аналогичным образом можно применять теорию индексов и для решения сравнений

$$ax^n \equiv b \bmod m \tag{7}$$

в случае составного модуля m, если для этого модуля существует хотя бы один первообразный корень, взяв который в качестве основания можно построить систему индексов.

Определение. Корнем n-й степени из класса a по модулю m называется класс чисел, удовлетворяющий сравнению

$$x^n \equiv a \bmod m. \tag{8}$$

Рассмотрим квадратные корни по любому модулю из единицы и определим их число. Начнем со случая, когда $m = p^k$, где $k \geq 1$, p – простое число.

Теорема. При любом нечетном простом p и $k \geq 1$ число квадратных корней из единицы по модулю p^k равно 2. При $p = 2$ число квадратных корней из единицы по модулю 2^k равно соответственно: 1 при $k = 0$ и 1; 2 при $k = 2$; 4 при $k \geq 3$.

Теорема. Пусть $m = 2^k p_1^{k_1} \dots p_s^{k_s}$, где $p_i > 2$ – каноническое разложение модуля m; тогда число квадратных корней из единицы по этому модулю равно:

1) 2^s при $k = 0$ или $k = 1$,
2) 2^{s+1} при $k = 2$,
3) 2^{s+2} при $k \geq 3$.

Доказательство. Нам надо определить число решений сравнения $x^2 \equiv 1 \bmod m$, которое эквивалентно системе:

$$\left.\begin{aligned} x &\equiv 1 \bmod 2^k \\ x^2 &\equiv 1 \bmod p_1^{k_1} \\ x^2 &\equiv 1 \bmod p_s^{k_s} \end{aligned}\right\}. \qquad (9)$$

Согласно предыдущей теореме каждое из этих сравнений, кроме первого, имеет два решения. Если обозначить через η число решений первого сравнения, то система (9), а следовательно, и сравнение $x^2 \equiv 1 \bmod m$, имеет $\eta 2^s$ решений.

В предыдущей теореме было определено, что: 1) при $k = 0$ и $k = 1$ имеем $\eta = 1$; 2) при $k = 2$ имеем $\eta = 2$; 3) при $k \geq 2$ имеем $\eta = 2^2$. Число квадратных корней из единицы по модулю m в этих трех случаях получается соответственно равным 2^s, 2^{s+1} и 2^{s+2}.

Примечание. При $m > 2$ каждому квадратному корню r из единицы по модулю m можно сопоставить отличный от r квадратный корень $\sqrt{m-r}$. Действительно, если $r^2 \equiv 1 \bmod m$, то и $(m - r)^2 \equiv 1 \bmod m$, причем $\sqrt{m-r} \neq r$, так как если бы было $\sqrt{m-r} = r$, то имели бы $m - r \equiv r \bmod m$, $2r \equiv 0 \bmod m$. Вместе с тем из $r^2 \equiv 1 \bmod m$ видно, что $(m, r) = 1$, так что было бы $m|2$, что противоречит условию $m > 2$.

Пример. Определить число квадратных корней из единицы по модулю $m = 60$.

Поскольку $600 = 2^s \cdot 3 \cdot 5^2$, т. е. $k = 3$, s = 2, число таких корней будет равно $2^4 = 16$.

СРАВНЕНИЯ ВТОРОЙ СТЕПЕНИ ПО ПРОСТОМУ МОДУЛЮ И КВАДРАТИЧНЫЕ ВЫЧЕТЫ

Общий вид сравнения второй степени по простому модулю p имеет вид

$$c_0x^2 + c_1x + c_2 \equiv 0 \bmod p. \qquad (1)$$

Поиск решения сравнения (1) при $p = 2$ не представляет труда, так как в этом случае оно принадлежит классу [0] или [1]. Изучать (1) будем в предположениях, что модуль p – простое нечетное число, которое не делит коэффициент c_0. Если $p|c_0$, то сравнение второй степени эквивалентно сравнению первой степени $c_1x + c_2 \equiv 0 \bmod p$.

Теорема. Если $p > 2$ и p не делит c_0 , то сравнение (1) эквивалентно сравнению вида

$$(x + c)^2 \equiv a \bmod p. \qquad (2)$$

Доказательство. Сравнение (1) (если p не делит c_0) можно сначала привести к виду

$$x^2 + b_1x + b_2 \equiv 0 \bmod p. \qquad (3)$$

Рассмотрим два возможных случая. Если $2|b_1$ (2 делит b_1), то сравнение (3) можно записать в виде

$$\left(x + \frac{b_1}{2}\right)^2 \equiv \left(\frac{b_1}{2}\right)^2 - b_2 \bmod p\,.$$

Если 2 не делит b_1, то сравнение (3) заменяем эквивалентным сравнением

$$x^2 + (b_1 + p)x + b_2 \equiv 0 \bmod p,$$

которое, поскольку p нечетно и, следовательно, 2 делит $b_1 + p$, можно записать в виде

$$\left(x + \frac{b_1 + p}{2}\right)^2 \equiv \left(\frac{b_1 + p}{2}\right)^2 - b_2 \bmod p\,.$$

Доказательство теоремы завершено.

Если положить $x + c = z$, то исследование сравнения (1) сводится к исследованию сравнения $z^2 \equiv a \bmod p$. Итак, рассмотрим сравнение

$$x^2 \equiv a \bmod p. \tag{4}$$

Будем считать, что модуль p не делит a, так как в этом случае сравнение (4) имеет только одно решение $x \equiv 0 \bmod p$.

Если p не делит a, то сравнение (4) может не иметь решений. Например, легко проверить, что сравнению $x^2 \equiv 2 \bmod 5$ не удовлетворяет ни один из классов по модулю 5.

Теорема. Сравнение (4) по простому модулю может иметь не больше, чем два решения.

Таким образом, если исключить из рассмотрения значения a, кратные p, то сравнение (4) либо имеет два решения, либо не имеет ни одного решения. Если при некотором a сравнение (4) имеет два решения, то и для любого $b \equiv a \bmod p$ это сравнение имеет два решения, поэтому естественно рассматривать сравнение (4) не для отдельных чисел a, а для соответствующих классов.

Определение. Класс чисел по модулю p называется классом квадратичных вычетов по этому модулю, если для чисел a, принадлежащих этому классу, сравнение $x^2 \equiv a \bmod p$ имеет два решения.

Определение. Класс чисел по модулю p называется классом квадратичных невычетов, если для чисел a, принадлежащих этому классу, сравнение $x^2 \equiv a \bmod p$ не имеет решений.

Все числа, принадлежащие классам квадратичных вычетов, т. е. все числа a, для которых сравнение

$$x^2 \equiv a \bmod p$$

имеет два решения, будем называть квадратичными вычетами по модулю p и аналогично все числа a, для которых это сравнение не имеет решений, – квадратичными невычетами по этому модулю. Числа a, принадлежащие классу $\bar{0}$, для которых сравнение (4) имеет одно решение, не причисляются ни к квадратичным вычетам, ни к квадратичным невычетам.

Теорема. Необходимым и достаточным условием того, чтобы a было квадратичным вычетом по простому модулю p ($p > 2$, p не делит a), является справедливость сравнения $a^{(p\sim1)/2} = 1 \bmod p$.

Доказательство [3]. Согласно теореме Ферма имеем $a^{p\sim1} = 1 \bmod p$. Тогда

$$a^{p\sim1} - 1 = (\mathrm{a}^{(p\sim1)/2} \sim 1)\,(\mathrm{a}^{(p\sim1)/2} + 1) \equiv 0 \bmod p$$

Оба сомножителя в последнем сравнении на модуль p делиться не могут, так как их разность, которая равна 2, делилось бы на p. Значит, для каждого целого a ($1 < a < p$) выполняется одно из сравнений:

$$a^{(p\sim1)/2} = 1 \bmod p,$$

$$a^{(p\sim1)/2} = \sim1 \bmod p$$

Если a – квадратичный вычет по модулю p, то существует такое число c, что $a = c^2 \bmod p$. Возводя обе части последнего сравнения в степень $(\tilde{p}1)/2$, получаем $a^{(p\sim1)/2} = c^{31}$. Но по теореме Ферма $c^{3\sim1} = 1$. Окончательно получаем, что для любого квадратичного вычета справедливо равенство

$$a^{(p\sim1)/2} = 1 \bmod p.$$

Теорема. Необходимым и достаточным условием того, чтобы a было квадратичным невычетом по простому модулю p ($p > 2$, p не делит a), является выполнение соотношения

$$a^{(p\sim1)/2} = \sim1 \bmod p. \qquad (7)$$

Доказательство. Согласно теореме Ферма имеем

$$a^{p\sim1} - 1 = (a^{(p\sim1)/2} \sim 1)\,(a^{(p\sim1)/2} + 1) \equiv 0 \bmod p. \qquad (8)$$

Если a – квадратичный невычет по модулю p, то первый множитель не делится на p и, следовательно, на p делится второй множитель, т. е. выполнено условие (7). Оба множителя в (8) не могут одновременно делиться на p, так как тогда их разность 2 тоже делилась бы на $p > 2$, что невозможно. Поэтому, если выполнено условие (7), то $a^{(p\sim1)/2} = \sim1 \bmod p$, т. е. a – квадратичный невычет по модулю p.

Последние две теоремы можно объединить в виде критерия, данного впервые Эйлером. Этот критерий позволяет судить, является ли целое число a, не делящееся на простой модуль $p > 2$, квадратным вычетом или невычетом по этому модулю.

Критерий Эйлера. Число a, не делящееся на простое число p, ($p > 2$), является квадратичным вычетом или невычетом по модулю p в зависимости от того, будет ли $a^{(p-1)/2}$ сравнимо $c + 1$ или $c - 1$ по этому модулю p.

Примеры. Число 2 – квадратичный невычет по простому модулю 11, так как $2^5 = 32 \equiv -1 \bmod 11$. Число 3 – квадратичный вычет по модулю 11, так как $3^5 = 43 \equiv 1 \bmod 11$.

Теорема. По любому простому модулю $p > 2$ число классов квадратичных вычетов равно числу классов квадратичных невычетов.

Теорема 6. Числа

$$1^2, 2^2, \ldots, \left(\frac{p-1}{2}\right)^2$$

образуют систему представителей всех классов квадратичных вычетов по простому модулю $p > 2$.

Пример. Найти классы квадратичных вычетов по модулю $p = 17$.

Берем числа

$$1^2 = 1,\ 2^2 = 4,\ 3^2 = 9,\ 4^2 = 16,$$

$$5^2 \equiv 8 \bmod 17,\ 6^2 \equiv 2 \bmod 17,$$

$$7^2 \equiv 15 \bmod 17,\ 8^2 \equiv 13 \bmod 17.$$

Классами квадратичных вычетов по модулю 17 являются классы

$$[1],\ [2],\ [4],\ [8],\ [9],\ [13],\ [15],\ [16].$$

Пример. Рассмотрим число $p = 7$. Тогда

$$1^2 = 1 \equiv 1 \bmod 7,$$

$$2^2 = 4 \equiv 4 \bmod 7,$$

$$3^2 = 9 \equiv 2 \bmod 7,$$

$$4^2 = 16 \equiv 2 \bmod 7,$$

$$5^2 = 25 \equiv 4 \bmod 7,$$

$$6^2 = 36 \equiv 1 \bmod 7.$$

Из приведенных соотношений видно, что для $p = 7$ только числа 1, 2, 4 выступают представителями класса квадратичных вычетов. Эти вычеты определяются уже для чисел 1^2, 2^2, 3^2. Заметим, что числа 3, 5, 6 являются квадратичными невычетами, так что уравнения

$$x^2 \equiv 3 \bmod 7,$$

$$x^2 \equiv 5 \bmod 7,$$

$$x^2 \equiv 6 \bmod 7$$

не имеют решения.

При изучении сравнений второй степени удобно пользоваться так называемым символом Лежандра. Введение этого символа, как будет видно из дальнейшего, значительно упрощает запись многих результатов и облегчает вычисления. Символ Лежандра для числа a по простому модулю $p > 2$ принято записывать в виде $\left(\frac{a}{p}\right)$ или в виде $L(a, b)$, причем этот символ определяется следующим образом.

Определение. Пусть p – простое число, $p > 2$ и p не делит a. Символом Лежандра $L(a, b)$ обозначается + 1 или – 1, смотря по тому, будет ли a квадратичным вычетом или невычетом по модулю p, т. е. 0, если $a \equiv 0 \bmod p$;

$L(a, p) = 1$, если a – квадратичный вычет по модулю p,

-1, если a – квадратичный невычет по модулю p.

Другими словами, $L(a, b)$ равно + 1, если сравнение $x^2 \equiv a \bmod p$ имеет два решения, и $L(a, b)$ равно -1, если это сравнение не имеет решений.

Примеры. $L(3, 11) = 1$, так как сравнение $x^2 \equiv 3 \bmod 11$ имеет два решения: $x \equiv \pm 5 \bmod 11$; $L(2, 5) = -1$, так как сравнение $x^2 \equiv 2 \bmod 11$ не имеет решений.

Теорема. Если $a \equiv b \bmod p$, то $L(a, p) = L(b, p)$.

Доказательство. Если $L(a, p) = 1$, т. е. a – квадратичный вычет по модулю p, то и любое $b \in [a]$ тоже будет квадратичным вычетом по этому модулю, и $L(b, p) = 1$. Если $L(a, p) = -1$, то и весь класс $[a]$ состоит из квадратичных невычетов по модулю p, т. е. при $a \equiv b \bmod p$, $L(b, p) = -1$.

Приведем основные свойства символа Лежандра.

1. Если $a \equiv b \bmod p$, то $L(a, p) = L(b, p)$.
2. $L(a_1 \times a_2 \times \ldots \times a_k) = L(a_1, p) \times L(a_2, p) \times \ldots \times L(a_k, p)$.
3. $L(a^2, p) = 1$.
4. $L(1, p) = 1$.
5. $L(a, p) = a^{(p-1)/2} \bmod p$ (критерий Эйлера).
6. $L(-1, p) = (-1)^{(p-1)/2} \bmod p$.
7. $L(2, p) = (-1)^{(p^2-1)/8}$
8. $L(q, p) = -1^{[(p-1)/2][(q-1)/2]} L(p, q)$ – закон взаимности. Этот закон можно записать в виде

$$L(q, p)L(p, q) = -1^{[(p-1)/2][(q-1)/2]}.$$

На основании перечисленных свойств заметим, что если $a = a_1^{\alpha 1} \ldots a_k^{\alpha k}$, то $L(a, p) = L(a_1, p)^{\alpha 1} \ldots L(a_k, p)^{\alpha k}$. В данном равенстве, учитывая свойство 3, можно отбросить множители, у которых показатель α_j, $j \le k$ является четным числом. Если же некоторое $a_j = 2$, $j \le k$, то свойство 7 позволяет вычислить значение

$$L(a_j, p)^{\alpha j} = L(2, p)^{\alpha j}.$$

Вышесказанное дает возможность сформулировать алгоритм вычисления символа Лежандра.

1. Если число a отрицательное, то выделяем множитель $L(-1, p)$.
2. Заменяем число a на остаток от деления числа a на p.
3. Раскладываем число a в произведение простых сомножителей

$$a = a_1^{\alpha 1} a_2^{\alpha 2} \ldots a_k^{\alpha k}.$$

Если число на простые множители не разлагается, то переходим на шаг 7.

4. Переходим к разложению

$$L(a, p) = L(a_1, p)^{\alpha 1} \ldots L(a_k, p)^{\alpha k}.$$

5. Отбрасываем множители с четным значением показателя $\alpha_j, j \le k$.
6. Вычисляем символ Лежандра $L(a_j, p)^{\alpha j}$ для $a_j = 2$.
7. Если все символы Лежандра в выражении

$$L(a, p) = L(a_1, p)^{\alpha 1} \ldots L(a_k, p)^{\alpha k}$$

известны, то вычисления $L(a, p)$ завершаются. В противном случае для множителей $L(a_j, p)^{\alpha j}$, у которых $a_j \ne 2$, применяем закон взаимности

$$L(p, q) = -1^{[(p-1)/2][(q-1)/2]} L(q, p).$$

В формуле для $L(p, q)$ полагается $p = a_j$.

8. Переходим к шагу 1.

Замечание. На каждом шаге процесс вычисления символа Лежандра может быть завершен. Это произойдет в том случае, если на каком-то шаге алгоритма все числа $\alpha_j, j \le k$ окажутся равными 1, 2 или –1.

Пример. Вычислить $L(68, 113)$. Здесь $a = 68$, а $p = 113$ есть простое число.

Алгоритм вычисления $L(68, 113)$

1. Так как $a > 0$, то множитель $L(-1, 113)$ отсутствует.
2. Так как $68 < 113$, то остаток равен самому числую
3. Разлагаем число $a = 68$ на простые множители $68 = 2^2 \times 17$.
4. Имеем

$$L(68, 113) = L(2^2 \times 17, 113) = L(2^2, 113) L(17, 113) = \\ = [L(2, 113)]^2 L(17, 113)$$

5. Так как $L(2^2, 113) = 1$ (см. свойство 3), то в разложении

$$L(68, 113) = [L(2, 113)]^2 L(17, 113)$$

отбрасываем множитель $[L(2, 113)]^2$.

6. После выполнения шага 5 получаем

$$L(68, 113) = L(17, 113).$$

В полученном разложении отсутствует множитель $L(2, 113)$.

7. Для множителя $L(17, 113)$, применяя закон взаимности для $a = 17$ и $p = 113$, получаем:

$$L(17, 113) = (-1)^{8 \times 56} L(113, 17) = L(113, 17).$$

8. В итоге получаем $L(68, 113) = L(113, 17)$.

Далее переходим на шаг 1 и начинаем вычислять $L(113, 17)$, здесь $a = 113$, а $p = 17$ есть простое число.

Алгоритм вычисления $L(113, 17)$

1. Так как $a > 0$, то множитель $L(-1, 17)$ отсутствует.

2. Остаток от деления числа 113 на 17 равен 11, поэтому получаем

$$L(113, 17) = L(11, 17).$$

3. Число 11 – простое и на множители не разлагается.

4. Так как 11 число простое, то переходим к шагу 7.

7. Для множителя $L(11, 17)$, применяя закон взаимности для $a = 11$ и $p = 17$, получаем

$$L(11, 17) = (-1)^{5 \times 8} L(17, 11) = L(17, 11).$$

8. В итоге имеем

$$L(113, 17) = L(17, 11).$$

Далее переходим на шаг 1 и начинаем вычислять $L(17, 11)$, здесь $a = 17$, $p = 11$ – простое число.

Алгоритм вычисления $L(17, 11)$.

1. Так как $a > 0$, то множитель $L(-1, 11)$ отсутствует.

2. Остаток от деления числа 17 на 11 равен 6.

3. Разлагаем число 6 на простые множители $6 = 2 \times 3$.

4. Имеем $L(6, 11) = L(2,11)L(3,11)$.

5. В разложении множители с четным значением показателей отсутствуют, поэтому переходим к шагу 6.

6. Вычисляем значение $L(2,11)$:

$$L(2,11) = [-1]^{(\alpha - 1)/8} = [-1]^{15} = -1,$$

где $\alpha = p^2 = 11^2 = 121$.

7. Для множителя $L(3, 11)$, применяя закон взаимности для $a = 3$ и $p = 11$, получаем

$$L(3, 11) = (-1)^{1 \times 5} L(11, 3) = -L(11, 3).$$

8. В итоге получаем

$$L(17, 11) = (-1) \times [-L(11, 3)] = L(11,3).$$

Далее переходим на шаг 1 и начинаем вычислять $L(11, 3)$, здесь $a = 11$, $p = 3$ – простое число.

Алгоритм вычисления $L(11, 3)$

1. Так как $a > 0$, то множитель $L(-1, 3)$ отсутствует.
2. Остаток от деления числа 11 на 3 равен 2, поэтому получаем:

$$L(11, 3) = L(2, 3).$$

3. Число 2 – простое, на множители не разлагается.
4. Имеем $L(11, 3) = L(2,3)$.
5. В разложении множители с четным значением показателей отсутствуют, поэтому переходим к шагу 6.
6. Вычисляем значение $L(2,3)$

$$L(2, 3) = [-1]^{(\alpha - 1)/8} = [-1]^1 = -1,$$

где $\alpha = p^2 = 3^2 = 9$.

Окончательно получаем, что $L(68, 113) = -1$.

Тем самым вычисление символа Лежандра $L(68, 113)$ завершено. Значение символа Лежандра $L(68, 13) = -1$ позволяет сделать вывод, что сравнение

$$x^2 \equiv 68 \bmod 113$$

не имеет решений.

Определение. Пусть нечетное число p имеет следующее разложение на простые множители $p = p_1 p_2 \ldots p_k$, где p_j, $j \le k$ – простые числа, среди которых могут быть одинаковые. Символ Якоби $J(a, p)$ определяется равенством

$$J(a, p) = L(a, p_1)\, L(a, p_2) \ldots L(a, p_k).$$

Заметим, что символ Лежандра является частным случаем символа Якоби. Более того, если число p простое, то символ Якоби по определению

является символом Лежандра. Вместе с тем символ Якоби $J(a, p)$ может равняться 1, т. е. $J(a, p) = 1$, а сравнение $x^2 \equiv a \bmod p$ не имеет решений.

Пример. Определим символ Якоби $J(2, 15)$. Знаем, что

$$J(2, 15) = L(2,3)L(2,5).$$

Вычислим $L(2, 3)$

$$L(2, 3) = [-1]^{(\alpha - 1)/8} = [-1]^1 = -1 \ (\alpha = p^2 = 3^2 = 9)$$

и $L(2, 5)$

$$L(2, 5) = [-1]^{(\alpha - 1)/8} = [-1]^3 = -1 \ (\alpha = p^2 = 5^2 = 25).$$

Окончательно получаем

$$J(2, 15) = L(2,3)L(2,5) = (-1)(-1) = 1.$$

Итак, $J(2, 15) = 1$, а сравнение $x^2 \equiv 2 \bmod 15$ не имеет решений.

Приведем основные свойства символа Якоби.

1. Если $a \equiv b \bmod p$, то $J(a, p) = J(b, p)$.
2. $J(a_1 \times a_2 \times \ldots \times a_k) = J(a_1, p) \times J(a_2, p) \times \ldots \times J(a_k, p)$.
3. $J(a^2, p) = 1$.
4. $J(1, p) = 1$.
5. $J(-1, p) = (-1)^{(p - 1)/2} \bmod p$.
6. $J(2, p) = (-1)^{(p^2 - 1)/8}$
7. $J(q, p) = -1^{[(p - 1)/2][(q - 1)/2]} L(p,q)$ – закон взаимности. Этот закон можно еще записать в виде $J(q, p)J(p, q) = -1^{[(p - 1)/2][(q - 1)/2]}$.

Алгоритм вычисления символа Якоби

1. Если число a отрицательно, то выделяем множитель $J(-1, p)$.
2. Заменяем число a на остаток от деления числа a на p
3. Если число a четно, то представляем его в виде $a = 2^t a_1$, где a_1 – нечетное число.
4. Переходим к разложению

$$J(a, p) = J(2, p)^t \, J(a_1, p).$$

5. Отбрасываем множитель $J(2, p)^t$, если t – четное число.
6. Если t нечетно, то вычисляем символ Лежандра $J(2, p)^t$.
7. Применяем к $J(q, p)$ закон взаимности

$$L(q, p) = -1^{[(p - 1)/2][(q - 1)/2]} L(p,q),$$

где полагается $q = a_1$.

8. Переходим к шагу 1.

Пример. Вычислить $J(506, 1\ 103)$, здесь $a = 506$, $p = 1\ 103$ – простое число. Так как 1 103 – простое число, то значение символа Якоби равно значению символу Лежандра, т. е. $J(506, 1\ 103) = L(506, 1\ 103)$.

Вычисление символа Якоби упрощается за счет того, что при составном числе a можно применять закон взаимности. При рассмотрении символов Лежандра составное число a надо раскладывать на произведение простых множителей, что является непростой задачей.

Алгоритм вычисления $J(506, 1\ 103)$

1. Так как $a > 0$, то множитель $J(-1, 1\ 103)$ отсутствует.
2. Так как $506 < 1\ 103$, то остаток равен самому числу.
3. Число $a = 506$ представляем в виде $506 = 2 \times 253$.
4. Переходим к разложению

$$J(506, 1\ 103) = J(2, 1\ 103)J(253, 1\ 103).$$

5. Показатель степени $t = 1$ множителя $J(2, 1\ 103)$ нечетный, поэтому переходим к шагу 6.
6. Вычисляем

$$J(2, 1\ 103) = [-1]^{(\alpha - 1)/8} = [-1]^{152\,076} = 1,$$

где $\alpha = p^2 = 1\ 103^2$

7. Для множителя $J(253, 1\ 103)$, применяя закон взаимности при $a = 253$ и $p = 1\ 103$, получаем

$$J(253, 1\ 103) = (-1)^{126 \times 551} J(1\ 103, 253) = J(1\ 103, 253).$$

8. В итоге имеем $J(506, 1\ 103) = J(1\ 103, 253)$.

Далее переходим на шаг 1 и начинаем вычислять $J(1\ 103, 253)$, здесь $a = 1103$, $p = 253$;

Алгоритм вычисления $J(1\ 103, 253)$

1. Так как $a > 0$, то множитель $J(-1, 253)$ отсутствует.
2. Остаток от деления числа 1103 на 253 равен 91, поэтому

$$J(1\ 103, 253) = J(91, 253).$$

3. Число $a = 91$ – нечетное, переходим на шаг 7.
4. Не выполняется.
5. Не выполняется.
6. Не выполняется.
7. Для символа Якоби $J(91,253)$, применяя закон взаимности для $a = 91$ и $p = 253$, получаем

$$J(91, 253) = (-1)^{45 \times 126} J(253,91) = J(253, 91).$$

8. В итоге имеем $J(1103, 253) = J(253, 91)$.

Далее переходим на шаг 1 и начинаем вычислять $J(253, 91)$, здесь $a = 253$, $p = 91$.

Алгоритм вычисления $J(253, 51)$

1. Так как $a > 0$, то множитель $J(-1, 91)$ отсутствует.
2. Остаток от деления числа 253 на 91 равен 71, поэтому

$$J(253, 91) = J(71, 91).$$

3. Число $a = 71$ – нечетное, переходим на шаг 7.
4. Не выполняется.
5. Не выполняется.
6. Не выполняется.
7. Для символа Якоби $J(71, 91)$, применяя закон взаимности для $a = 71$ и $p = 91$, имеем

$$J(71, 91) = (-1)^{35 \times 45} J(91, 71) = -J(91, 71).$$

8. В итоге получаем $J(253, 91) = -J(91, 71)$.

Далее переходим на шаг 1 и начинаем вычислять $J(91, 71)$, здесь $a = 91$, $p = 71$.

Алгоритм вычисления $J(91, 71)$

1. Так как $a = 91 > 0$, то множитель $J(-1, 71)$ отсутствует.
2. Остаток от деления числа 91 на 71 равен 20, поэтому

$$J(91, 71) = J(20, 71).$$

3. Число $a = 20$ представим в виде $a = 2^2 \times 5$.
4. Переходим к разложению

$$J(20, 71) = J(2, 71)^2 J(5, 71).$$

5. Показатель степень у множителя $J(2, 71)$ t=2 четный, поэтому

$$J(2, 71)^2 = 1.$$

Переходим к шагу 7.

6. Не выполняется.
7. Для символа Якоби $J(5, 71)$, применяя закон взаимности для $a = 5$ и $p = 71$. Имеем

$$J(5, 71) = (-1)^{2 \times 35} J(71, 5) = J(71, 5).$$

8. В итоге получаем $-J(91, 71) = -J(71, 5)$.

Далее переходим на шаг 1 и начинаем вычислять $-J(71, 5)$, где $a = 71$, $p = 5$.

Алгоритм вычисления $J(71, 5)$

1. Так как $a = 71 > 0$, то множитель $J(-1, 71)$ отсутствует.
2. Остаток от деления числа 71 на 5 равен 1, поэтому

$$J(71, 5) = J(1, 5).$$

3. Число $a = 1$ – нечетное, переходим на шаг 7.
4. Не выполняется.
5. Не выполняется.
6. Не выполняется.
7. Вычисляем символ Якоби $J(1, 5) = 1$.
8. В итоге получаем $-J(71, 5) = -1$.

Вычисление символа Якоби $J(1103,253)$ завершено. Окончательно получаем $J(1103, 253) = -1$. Значение символа Якоби $J(506,1103) = -1$ позволяет сделать вывод, что сравнение $x^2 \equiv 506 \bmod 1103$ не имеет решений.

ВЫЧИСЛЕНИЕ КВАДРАТНЫХ КОРНЕЙ ПО МОДУЛЮ

Случай простого модуля

Рассмотрим сравнение

$$x^2 \equiv a \bmod p, \qquad (1)$$

где число $p \neq 2$ простое и целое число a не делится на p.

Вычисление решения x данного уравнения является трудной задачей теории чисел и используется в криптографии с открытыми ключами для построения односторонней функции. Далее рассмотрим алгоритмы вычисления квадратных корней по целочисленному простому модулю.

Для того чтобы узнать, разрешимо ли сравнение, достаточно вычислить символ Лежандра. Если $L(a, p) = -1$, то сравнение не имеет решения. При $L(a, p) = 1$ сравнение разрешимо и имеет ровно два решения. Действительно, если $L(a, p) = 1$, то по определению символа Лежандра сравнение (1) имеет хотя бы одно решение $x_1 \bmod p$. Пусть x_2 – другое решение сравнения. Тогда $x_1^2 \equiv a \bmod p$, $x_2^2 \equiv a \bmod p$, т. е. $x_1^2 - x_2^2 \equiv 0 \bmod p$. Значит, что хотя бы одно из выражений $x_1 - x_2$, $x_1 + x_2$ должно делиться на p. В первом случае получаем уже имеющееся решение x_1, во втором – решение $x_2 = -x_1 \bmod p$. При этом значения x_1 и $-x_1$ различны, в противном случае выполнялось бы соотношение $2x_1 \equiv 0 \bmod p$, что невозможно, поскольку $p \neq 2$ НОД(x_1,p) = НОД(a,p) = 1. Кроме того, согласно сформулированным ранее теоремам сравнение (1) не может иметь более двух решений.

Рассмотрим некоторые способы решения сравнения (1) в зависимости от вида модуля [3], [14].

Пусть $p \equiv 3 \bmod 4$, т. е. $p = 4m + 3$, где m – целое число. Разрешимость сравнения означает, что $L(a, p) = 1$. По свойству символа Лежандра имеем $1 = L(a, p) \equiv a^{2m+1} \bmod p$. Тогда

$$(a^{m+1})^2 = a^{2m+2} = a^{2m+1}\, a \equiv a \bmod p.$$

Таким образом, решение имеет вид $x \equiv \pm a^{m+1} \bmod p$.

Пример [3]. Решим сравнение $x^2 = 7 \bmod 31$. Вычисляем символ Лежандра: $L(7, 31) = 1$, значит, сравнение разрешимо. Число 31 представляем в виде $31 = 4 \cdot 7 + 3$, т. е. $m = 7$. Находим решение:

$$x \equiv \pm 7^8 = \pm 7^3 7^3 7^2 \equiv \pm 2 \cdot 2 \cdot 18 = \pm 10 \bmod 31.$$

Проверка: $(\pm 10)^2 - 7 = 100 - 7 = 93 = 31 \cdot 3$.

Пусть $p \equiv 5 \bmod 8$, т. е. $p = 8m + 5$, где m – целое число. Разрешимость сравнения означает, что $L(a, p) = 1$. По свойству символа Лежандра получаем $1 = L(a, p) = a^{(p-1)/2} = a^{4m+2} \equiv (a^{2m+1})^2 \bmod p$. Отсюда $a^{2m+1} = 1 \bmod p$ или $a^{2m+1} = -1 \bmod p$. В первом случае, умножая обе части сравнения на a, получаем $a^{2m+1} = a \bmod p$, т. е. решение имеет вид $x \equiv a^{m+1} \bmod p$. При $a^{2m+1} \equiv -1 \bmod p$ ситуация немного сложнее. Заметим, что при $p \equiv 5 \bmod 8$, число 2 является квадратичным невычетом по модулю p. Действительно,

$$L(2, p) = (-1)^{(\alpha-1)/2} = (-1)^{8\cdot\beta/8} = (-1)^{\beta} = -1,$$

где $\alpha = p^2$, $\beta = 8m^2 + 10m + 3$.

Таким образом, $(2^{2m+1})^2 \equiv -1 \bmod p$. Тогда $a^{2m+1} \cdot (2^{2m+1})^2 \equiv 1 \bmod p$.

Умножая обе части этого сравнения на a, получаем решение сравнения (2.11): $x \equiv \pm a^{m+1} \cdot 2^{2m+1} \bmod p$. Здесь вместо числа 2 можно брать любой другой квадратичный невычет по модулю p.

Пример. Решим сравнение $x^2 \equiv 10 \bmod 53$.

Вычисляем символ Лежандра $L(10, 53) = 1$, значит, сравнение разрешимо. Число 53 представляем в виде $53 = 8 \cdot 6 + 5$, т. е. $m = 6$. Поскольку $10^{2\cdot 6+1} \equiv 1 \bmod 53$, находим решение: $x \equiv \pm 10^7 \equiv \pm 13 \bmod 53$.

Проверка: $(\pm 13)^2 - 10 = 169 - 10 = 159 = 53 \cdot 3$.

Пример. Решим сравнение $x^2 \equiv 11 \bmod 37$. Вычисляем символ Лежандра $L(11, 37) = 1$, значит, сравнение разрешимо. Число 37 представляем в виде $37 = 8 \cdot 4 + 5$, т. е. $m = 4$.

Поскольку $11^{2\cdot 4+1} \equiv -1 \bmod 37$, находим решение: $x \equiv \pm 11^5 \cdot 2^9 \equiv \equiv \pm 14 \bmod 37$.

Проверка: $(\pm 14)^2 - 11 = 196 \cdot 11 = 185 = 37 \cdot 5$.

Пусть $p \equiv 1 \bmod 8$. Представим p в виде $p = 2^k \cdot h + 1$, где $k \geq 3$, число h нечетное. Разрешимость сравнения означает, что $L(a, p) = 1$. По свойству символа Лежандра $1 = L(a, p) \equiv a^{(p-1)/2}$. Имеем $\beta = (p-1)/2 = 2^k h/2 = 2^{k-1} \cdot h$ отсюда $a^{\beta} \equiv 1 \bmod p$ и, извлекая квадратный корень, получаем

$$a^{\beta/2} \equiv \pm 1 \bmod p \text{ (где } \beta/2 = 2k - 2).$$

Пусть N – произвольный квадратичный невычет по модулю p, т. е. $-1 = L(N, p) \equiv N^{\beta} \bmod p$, где $\beta = 2^{k-1}h$. Тогда, при некотором целом $s_2 \geq 0$, получим

$$a^{2^{k-2}h} N^{2^{k-1}hs_2} \equiv 1 \bmod p.$$

Откуда, извлекая корень,

$$a^{2^{k-3}h} N^{2^{k-2}hs_2} \equiv \pm 1 \bmod p.$$

Далее, при некотором целом $s_3 \geq 0$, получим

$$a^{2^{k-3}h} N^{2^{k-2}hs_2} \equiv 1 \bmod p,$$

откуда

$$a^{2^{k-4}h} N^{2^{k-3}hs_2} \equiv \pm 1 \bmod p \text{ и т. д.}$$

Получив сравнение

$$a^h N^{2s_k} \equiv 1 \bmod p$$

для некоторого целого $s_k \geq 0$ и, умножив обе его части на a, получаем решение

$$x \equiv \pm a^{\frac{h+1}{2}} N^{s_k} \bmod p \cdot$$

Алгоритм 1. Решение сравнения $x^2 \equiv a \bmod p$, p – простое число [3].

Вход. Простое число $p \neq 2$; такие целые числа a и N, что $L(a, p) = L(N, p) = 1$.

Выход. Решение сравнения $x^2 = a \bmod p$.

1. Представить число p в виде $p = 2^k \cdot h + 1$, где число h нечетное.

2. Положить

$$a_1 = a^{(h+1)/2} \bmod p,\ a^2 = a^{-1} \bmod p,\ N_1 = N^h \bmod p,\ N_2 = 1,\ j = 0.$$

3. Для $i = 0, 1, ..., k - 2$ выполнять следующие действия.

3.1. Положить $b = a_1 N_2 \bmod p$.

3.2. Вычислить $c = a_2 b^2 \bmod p$.

3.3. Вычислить абсолютно наименьший вычет

$$d = c^{2^{k-2-i}} \bmod p \cdot$$

При $d = 1$ положить $j_i = 0$, при $d = -1$ положить $j_i = 1$.

3.4. Положить

$$N_2 \leftarrow N_2 N_1^{2^i j_i} \bmod p\,.$$

4. Результат: $\pm a_1 N_2 \bmod p$.

Пример. Решим сравнение $x^2 = 14 \bmod 193$ [3].

Вычисляем символ Лежандра $L(14, 193) = 1$, значит, сравнение разрешимо.

Выбираем $N = 5$, $L(5, 193) = -1$.
Находим представление $193 = 2^6 \cdot 3 + 1$, т. е. $k = 6$, $h = 3$.
Полагаем

$$a_1 = 14^2 \equiv 3 \bmod 193,\ a_2 = 14^{-1} \equiv 69 \bmod 193,$$
$$N_1 = 5^3 \equiv 125 \bmod 193,\ N_2 = 1,\ j = 0.$$

Результаты вычислений сведем в таблицу.

i	$b = a_1 N_2$	$c = a_2 b^2$	$d \equiv c^{2^{4-i}}$	j_i	N_2
0	3	42	–1	1	125
1	182	50	–1	1	158
2	88	112	1	0	158
3	88	112	–1	1	39
4	117	192	–1	1	122

Тогда решением сравнения будет $x = \pm 3 \cdot 122 = \pm 173 \bmod 193$.
Проверка: $(\pm 173)^2 - 14 = 29\,929 - 14 = 29\,915 = 193 \cdot 155$.

Случай составного модуля

Рассмотрим сравнение

$$x^2 \equiv a \bmod p$$

при условии, что модуль p является составным числом, т. е. $p = p_1 p_2 \ldots p_k$, где p_i – простые числа, $i = 1, 2, \ldots, k$. Сформулируем несколько теорем для этого случая [3].

Теорема. Пусть число p простое, $p \neq 2$, целое число a не делится на p и $n \in N$, $n \geq 1$. Для того чтобы сравнение $x^2 \equiv a \bmod p^n$ было разрешимо, необходимо и достаточно, чтобы было разрешимо сравнение $x^2 \equiv a \bmod p$.

Доказательство. Если сравнение разрешимо и x_n – его решение, т. е., $x_n^2 \equiv a \bmod p^n$, то разность $x_n^2 - 1$ a делится на p^n, а значит и на p, и x_n является решением сравнения $x^2 \equiv a \bmod p$.

Пусть теперь разрешимо сравнение $x^2 \equiv a \bmod p$ и $x \equiv x_1 \bmod p$ – его решение. Тогда $x = x_1 + pk_1$ для некоторого целого k_1. Подставляя это значение x в сравнение $x^2 \equiv a \bmod p^2$, получаем

$$x_1^2 + 2x_1 p k_1 = a \bmod p^2,$$

а так как разность $x_1^2 - a$ делится на p, получаем

$$2x_1 k_1 \equiv \frac{-(x_1^2 - a)}{p} \bmod p. \qquad (2.14)$$

Числа a и p взаимно просты, поэтому числа x_1 и p тоже взаимно просты; кроме того, $p \neq 2$. Поэтому НОД$(2x_1, p) = 1$, т. е., целое число $\frac{x_1^2 - a}{p}$ делится на НОД$(2x_1, p)$, в этом случае можно доказать [3], что сравнение (2.14) имеет единственное решение $k_1 \equiv k_1' \bmod p$, $k_1 = k_1' + pk_2$ для некоторого целого k_2. Тогда $x = x_1 + pk_1 = x_1 + pk_1' + p^2k_2 = x_2 + p^2k_2$. Подставляя это значение в сравнение $x^2 \equiv a \bmod p^3$, получаем

$$x_2^2 + 2x_2p^2k_2 \equiv a \bmod p^3,$$

$$2x_2k_2 \equiv \frac{-(x_2^2 - a)}{p^2} \bmod p. \qquad (2.15)$$

Здесь НОД(x_2, p) = НОД$(x_1 + pk_1, p)$ = НОД$(x_1, p) = 1$, поэтому сравнение (2.15) имеет единственное решение $k_2 \equiv k_2' \bmod p$, $k_2 = k_2' + pk_3$ для некоторого целого k_3. Тогда $x = x_2 + p^2 k_2 = x_2 + p^2 k_2' + p^3k_3 = x_3 + p^3k_3$.

Продолжая аналогичные рассуждения, получаем решение $x \equiv x_n \bmod p^n$ сравнения $x^2 \equiv a \bmod p$, где $x_n \equiv x_1 \bmod p$.

Второе решение этого сравнения имеет вид $-x_n \equiv -x_1 \bmod p$.

Заметим, что найденные решения определяют все решения сравнения $x^2 \equiv a \bmod p$ [3].

Теорема. Пусть $m = m_1m_2...m_t$, где числа m_1, m_2, ..., m_t попарно взаимно просты. Для того чтобы сравнение $x^2 \equiv a \bmod m$ было разрешимо, необходимо и достаточно, чтобы были разрешимы все сравнения

$$x^2 \equiv a \bmod m_1,$$

$$x^2 \equiv a \bmod m_2,$$

$$\ldots,$$

$$x^2 \equiv a \bmod m.$$

Доказательство. Если c – решение сравнения $x^2 \equiv a \bmod m$, то разность $c^2 - a$ делится на произведение $m_1m_2...m_t$, а значит, и на каждое из чисел $m_1m_2...m_t$.

Пусть теперь u_1, u_2, $...u_t$ – решения сравнений по соответствующим модулям, т. е. $u_i^2 \equiv a \bmod m_i$ для $i = 1, 2, ..., r$. Числа $m_1m_2...m_t$ попарно взаимно просты, значит, по китайской теореме об остатках, существует такое целое число u, что $u \equiv u_i \bmod m_i$ для всех $i = 1, 2, ..., r$. Тогда $u^2 \equiv u_i^2 \equiv a \bmod m_i$ для $i = 1,2, ...,r$, т. е. разность $u^2 - a$ делится на каждое из чисел $m_1m_2...m_t$,

а значит, по свойству 7 наибольшего общего делителя, и на произведение $m_1 m_2 ... m_t = m$.

Теорема. Пусть число a нечетное. Справедливы следующие утверждения:

1. Сравнение $x^2 \equiv a \bmod 2$ разрешимо при любом a.

2. Сравнение $x^2 \equiv a \bmod 2^2$ разрешимо тогда и только тогда, когда $a \equiv 1 \bmod 4$.

3. Сравнение $x^2 \equiv a \bmod 2^n$, где $n \geq 3$, разрешимо тогда и только тогда, когда $a \equiv 1 \bmod 8$.

Доказательство [3]. Если число a нечетное, то $a \equiv 1 \bmod 2$, и сравнение из п. 1 имеет вид $x^2 \equiv 1 \bmod 2$. Это сравнение имеет единственное решение то $x \equiv 1 \bmod 4$.

Если $a \equiv 1 \bmod 4$, т. е. $a = 4k + 1$, то сравнение вида

$$x^2 \equiv 4k + 1 \equiv 1 \bmod 4$$

имеет два решения: $x \equiv \pm 1 \bmod 4$. Обратно, пусть c – решение сравнения $x^2 \equiv a \bmod 4$. Нечетное число a имеет вид $a = 2k+ 1$, где k – целое число. Кроме того, так как число a – нечетное, то c тоже нечетное, $c = 21 + 1$. Отсюда

$$(21 + 1)^2 \equiv 2k + 1 \bmod 4,\ 0 \equiv 2k \bmod 4,$$

т. е. $k \equiv 0 \bmod 2$, и $a \equiv 1 \bmod 4$.

Осталось рассмотреть случай $n \geq 3$. Здесь решение тоже будет нечетным числом. Пусть $c = 2l + 1$ – решение сравнения. Аналогично предыдущему случаю получаем, $a \equiv 1 \bmod 4$, т. е., $a = 4k + 1$, где k – целое число.

Тогда

$$(2\ l + 1)^2 \equiv 4k + 1 \bmod 2^n,$$

$$4\ l^2 + 4l \equiv 4\ k \bmod 2^n,$$

$$l\ (l + 1) \equiv k \bmod 2^{n-2},$$

где $n \geq 3$. Произведение в левой части сравнения является четным числом, тогда и число k четное, и $a \equiv 1 \bmod 8$.

Пусть теперь $a \equiv 8k + 1$. При $n = 3$ решениями сравнения $x^2 \equiv$ a mod 8 будут $x \equiv \pm 1$ и $x \equiv \pm 3 \bmod 8$. Найдем решения сравнения $x^2 \equiv a \bmod 2^n$, для $n \geq 4$. Решение этого сравнения – нечетное число, т. е. сравнимо с 1 или с 3 по модулю 4 и имеет вид $x = \pm (1 + 4t_3)$, где $t_3 \in Z$.

Тогда

$$(1 + 4\ t_3)^2 \equiv 8k + 1 \bmod 2^n,$$

$$8\ t_3 + 16\ t_3^2 \equiv 8k \bmod 2^n,$$

в частности,

$$8\ t_3 \equiv 8k \bmod 2^4,$$

$$t_3 \equiv k \bmod 2,$$

таким образом $t_3 = t_3' + 2t_4$ для $t_3' \in \{0, 1\}$, t_4 – целое и

$$x = \pm(1 + 4\ t_3' + 8t_4) = \pm(x_4 + 8\ t_4).$$

Рассматривая сравнение по модулю 2^5, получаем

$$(x_4 + 8\ t_4)^2 \equiv 8\ k + 1 \bmod 2^5,$$

$$16\ k^2 + 16\ t_4 \equiv 0 \bmod 2^5,$$

$$t_4 \equiv k^2 \bmod 2,$$

таким образом $t_4 = t_4' + 2t_5$ для $t_4' \in \{0, 1\}$, t_5 – целое число, и

$$x = \pm(1 + 4t_3' + 8t_4' + 16t_5) = \pm(x_5 + 16t_5).$$

Продолжаем, пока не получим $x = \pm(x_n + 2^{n-1}t_n)$, где t_n – целое число. Целые числа с четным значением t_n дают два решения: $x \equiv \pm x_n \bmod 2^n$, целые числа с нечетным значением t_n – еще два решения $x \equiv \pm(x_n + 2^{n-1}) \bmod 2^n$. При $x \equiv x_n + 2^{n-1} \equiv 1 \bmod 4$, $-x \equiv -(x_n + 2^{n-1}) = 3 \bmod 4$.

Теорема. Пусть $m = m_1 m_2 \ldots m_t$, где числа $m_1 m_2 \ldots m_t$ попарно взаимно просты. Для того чтобы сравнение (2.7) было разрешимо, необходимо и достаточно, чтобы были разрешимы все сравнения

$$x^2 \equiv a \bmod m_1,\ x^2 \equiv a \bmod m_2, \ldots,\ x^2 \equiv a \bmod m_t.$$

Доказательство. Если c – решение сравнения (2.7), то разность $c^2 - a$ делится на произведение $m_1 m_2 \ldots m_t$, а значит, и на каждое из чисел $m_1 m_2 \ldots m_t$.

Пусть теперь $u_1, u_2, \ldots u_t$ – решения сравнений по соответствующим модулям, т. е. $u_i^2 \equiv a \bmod m_i$ для $i = 1, 2, \ldots, r$. Числа $m_1 m_2 \ldots m_t$ попарно вза-

имно просты, значит, по китайской теореме об остатках, существует такое целое число u, что $u \equiv u_i \bmod m_i$ для всех $i = 1, 2, ..., r$.

Тогда $u^2 \equiv u_i^2 \equiv a \bmod m_i$ для $i = 1, 2, ..., r$, т. е. разность $u^2 - a$ делится на каждое из чисел $m_1 m_2 ... m_t$, а значит, по свойству 7 наибольшего общего делителя, и на произведение $m_1 m_2 ... m_t = m$.

Следствие 1. Пусть

$$m = 2^n p_1^{\alpha_1} p_2^{\alpha_2} ... p_s^{\alpha_s}$$

каноническое разложение целого числа m и целое число a взаимно просто с m. Сравнение $x^2 \equiv a \bmod m$ разрешимо тогда и только тогда, когда

$a \equiv 1 \bmod 4$ при n=2, $a \equiv 1 \bmod 8$ при $n \geq 3$ и $L(a, p_1)$, $L(a, p_2)$, …, $L(a, p_s) = 1$. Это сравнение имеет

- 2^s решений при $n = 0$ или $n = 1$;
- 2^{s+1} решений при $n = 2$;
- 2^{s+2} решений при $n \geq 3$.

Вычисление квадратных корней по составному модулю

Из приведенной выше теории следует, что если $n = pq$, где p и q – простые числа, группа $Z_n^{\bullet}$ изоморфна пространству $Z_p^{\bullet} \times Z_q^{\bullet}$. Поскольку изоморфизм сохраняет свойства арифметических операций, то отношение

$$x^2 \equiv y \bmod n$$

выполняется тогда и только тогда, когда оно справедливо как по модулю p, так и по модулю q. Значит, если известно разложение числа n на множители, квадратный корень по модулю n можно вычислить с помощью следующего алгоритма.

Алгоритм 6.5. Квадратный корень по составному модулю

Ввод: простые числа p, q, удовлетворяющие условию $n = pq$;

целое число $y \in QR_n$.

Вывод: квадратный корень числа y по модулю n.

1. $x_p \leftarrow \sqrt{y \bmod p}$;

$x_q \leftarrow \sqrt{y \bmod q}$; (Применяем алгоритмы 6.3 или 6.4.)

2. return $(\overline{1}_p x_p + \overline{1}_p x_p \bmod n)$ (Применяем алгоритм 6.1.)

Число $y \bmod n$ имеет два разных квадратных корня, которые мы обозначим как x_p и $p - x_p$ соответственно. Аналогично обозначим квадратные корни числа $y \bmod q$ как x_q и $q - x_q$ соответственно. Вследствие изомор-

физма между группой $Z_n^\bullet$ и пространством $Z_p^\bullet \times Z_q^\bullet$ число $y \in QR_n$. имеет в группе $Z_n^\bullet$ ровно четыре квадратных корня, которые можно вычислить с помощью алгоритма 6.5

$$\left.\begin{aligned} x_1 &= \overline{1}_p x_p + \overline{1}_q x_q \\ x_2 &= \overline{1}_p x_p + \overline{1}_q x_q (q - x) \\ x_3 &= \overline{1}_p (p - x_p) + \overline{1}_q x_q \\ x_4 &= \overline{1}_p (p - x_p) + \overline{1}_q (q - x_q) \end{aligned}\right\} \bmod n. \qquad (1)$$

Применяя формулу (1) на шаге 2 алгоритма 6.5, можно вычислить все четыре квадратных корня исходного числа.

В качестве упражнения поставим следующую задачу: сколько разных квадратных корней имеет число $y \in QR_n$, если $n = pqr$, где p, q и r – разные простые числа?

Итак, если известно разложение числа n на множители, существует эффективный алгоритм, позволяющий вычислить квадратные корни любого заданного числа их множества QR_n. Что делать, если разложение числа n на множители неизвестно? На этот вопрос дает ответ третья часть следующей теоремы.

Теорема. Пусть pq, где p и q – разные нечетные простые числа и $y \in QR_n$. Тогда четыре корня числа y, вычисленные по формуле (1), обладают следующими свойствами.

1. Все они отличаются друг от друга.
2. $x_1 + x_2 + x_3 + x_4 = n$.
3. НОД$(x_1 + x_2, n)$ = НОД$(x_3 + x_4, n) = q$,
   НОД$(x_1 + x_3, n)$ = НОД$(x_2 + x_4, n) = p$.

Доказательство

1. Учитывая смысл обозначений $\overline{1}_p$ и $\overline{1}_q$, приходим к выводу, что, например, $x^1 \bmod q = x^q$ и $x^2 \bmod q = q - x^q$. Напомним, что x^q и $q - x^q$ – два разных квадратных корня числа $y \bmod n$. Из условия, что $x_1 \neq x_2 \bmod q$, следует, что $x_1 \neq x_2 \bmod n$, т. е. числа x^1 и x^2 отличаются друг от друга. Другие варианты доказываются аналогично.

2. Из формулы (1) следует, что

$$x_1 + x_2 = x_2 + x_3 = \overline{1}_p p + \overline{1}_q q.$$

Правая часть этого уравнения сравнима с нулем по модулю p и по модулю q. Поскольку эти корни принадлежат группе $Z_n^\bullet$, выполняется условие $0 < x^1 + x^4 = x^2 + x^3 < 2n$. Очевидно, что число n – единственное число в интервале $(0, 2n)$, сравнимое с нулем по модулю p и по модулю q. Значит,

$x^1 = n - x^4$ и $x^2 = n - x^3$. Рассмотрим лишь число $x^1 + x^2$. Другие варианты доказываются аналогично. Исследуя формулу (1), приходим к выводу, что

$$x^1 + x^2 = 2 \cdot \overline{1_p x_p + \bar{1}_q q}.$$

Таким образом, $x^1 + x^2 \bmod p \equiv 2x_p \neq 0$ и $x^1 + x^2 \equiv 0 \bmod q$. Значит, число $x^1 + x^2$ не равно нулю и кратно числу q, но не кратно числу p. Отсюда следует, что $\gcd(x_1 + x_2, n) = q$. Допустим, что существует алгоритм A, получающий на вход число $y \in QR_n$ и вычисляющий число x, удовлетворяющее условию $x^2 \equiv y \bmod n$. Тогда, для вычисления квадратного корня x числа x^2 можно выполнить алгоритм $A(x^2, n)$. Вероятность события $0 < \gcd(x + x^1, n) < n$ равна 0,5 (вероятностное пространство состоит из четырех квадратных корней числа y). Значит, алгоритм A эффективно выполняет факторизацию числа n.

Следствие 1. Пусть $n = pq$, где p и q – разные нечетные простые числа. Тогда факторизация числа n эквивалентна вычислению квадратного корня по модулю n.

Следствие 2. Пусть $n = pq$, где p и q – разные нечетные простые числа. Тогда для любого числа $y \in QR_n$ два квадратных корня числа y меньше $n/2$, а другие два корня – больше $n/2$.

Далее рассмотрим криптографическую систему, которая базируется на трудности решения сравнения $x^2 \equiv a \bmod p$.

ЦИФРОВАЯ ПОДПИСЬ ФИАТА – ШАМИРА

Рассмотрим подход к построению схемы цифровой подписи, основанной на сложности задач факторизации больших целых чисел и извлечения квадратного корня в кольце вычетов. Идея построения схемы принадлежит А. Фиату и А. Шамиру. Приведем одну из модификаций схемы, в которой реализуется цифровая подпись.

Пусть h – некоторая хеш–функция, преобразующая исходное сообщение в битовую строку длины m в битовую строку фиксированной длины. Выберем различные простые числа p и q и положим $n = pq$. В качестве секретного ключа каждый абонент должен сгенерировать m различных случайных чисел $a_1, a_2, \ldots, a_m \in Z_n$. Открытым ключом объявляется набор чисел $b_1, b_2, \ldots, b_m \in Z_n$, где $b_i = (a_i^{-1})^2 \bmod n$, $i = 1,\ldots, m$.

Алгоритм вычисления цифровой подписи для сообщения M состоит в выполнении следующих действий:

1) выбрать случайное число r, $1 \le r \le n - 1$;

2) вычислить $u = r^2 \bmod n$;

3) вычислить

$$h(M, u) = s = (s_1, s_2,\ldots, s_m);$$

4) вычислить

$$t = r\prod_{i=1}^{m} \bmod n;$$

5) подписью для сообщения M положить пару (s, t).

Алгоритм проверки подписи состоит в выполнении следующих действий:

1) по открытому ключу $b_1, b_2, \ldots, b_m \bmod n$ и значению t вычислить

$$w = t^2\prod_{i=1}^{m} b^{s_i} \bmod n;$$

2) вычислить $h(M, w) = s'$;

3) проверить равенство $s = s'$.

Достоинствами описанной схемы являются возможность выработки цифровых подписей для нескольких различных сообщений с использованием одного секретного ключа, а также сравнительная простота алгоритмов вычисления и проверки подписи. Например, для схемы цифровой под-

писи, основанной на алгоритме *RSA*, соответствующие алгоритмы требуют выполнения значительно большего числа умножений. Попытка компрометации этой схемы сталкивается с необходимостью решения сложной задачи нахождения квадратных корней по модулю n.

Недостатком схемы является большая длина ключа, которая определяется числом m. Если двоичная запись числа n содержит l знаков, то длина секретного ключа составляет ml бит, а открытого ключа – $(m + 1)l$ бит. При этом необходимо учитывать, что для обеспечения достаточной стойкости данной схемы цифровой подписи числа l и m должны иметь в своей двоичной записи несколько сотен бит.

ПРОСТЫЕ ЧИСЛА

Натуральное число p, больше единицы называется простым, если оно делится нацело только на 1 и на себя.

Теорема (Эвклид). Множество простых чисел бесконечно.

Обозначим через $\pi(x)$ функцию, которая равна числу простых чисел p в интервале $1 < x \le p$. Российский математик П. Л. Чебышев в 1850 году показал [7], что $0{,}921\, x / \ln x < \pi(x) < 1{,}106\, x / \ln x$.

Простые числа являются важным понятием в криптографии. Многие современные криптографические системы строятся на базе простого числа. Поэтому алгоритмы генерации простых чисел и проверки на простоту сформированного числа являются важными инструментами при создании криптографической системы.

Простые числа встречаются довольно часто. Заметим, что существует около 10^{151} простых чисел длиной от 1 до 512 бит включительно [10], а количество простых чисел меньших 2^{512} приблизительно равно 2^{503} [4]. Для чисел близких n вероятность произвольно выбранному числу оказаться простым числом, равна $1/\ln n$. При случайном выборе двух простых чисел в диапазоне от 1 до 151 бита вероятность совпадения этих чисел ничтожно мала. Простые числа играют важную роль в современной криптографии. Многие современные криптографические системы с открытыми (или не симметричными) ключами формируются с применением простых чисел.

Для простых чисел будем рассматривать три задачи:

- построение простых чисел;
- проверка чисел на простоту;
- факторизация (разложения) чисел на простые множители.

На самом деле все эти три задачи фактически дают ответ на один вопрос: является ли рассматриваемое число простым. Но для каждой из этих задач применяются свои методы.

Для первой задачи, используя необходимые условия простоты, можно давать ответы типа:

- заданное число n не простое;
- вероятность того, что заданное число n не простое, меньше заданного числа ε.

Для второй задачи можно строить некоторую последовательность чисел специального вида. И для чисел данной последовательности применять некоторые тесты до тех пор, пока не найдем среди них простое число.

Приведем некоторые определения, теоремы, алгоритмы, которые связаны с вопросами решения поставленных задач (см., например [3–5]; [7]; [10–11]).

Определение 1. Числа $F_k = 2^{\alpha} + 1$, $\alpha = 2^k$, $k = 0, 1, 2, \ldots$, называются числами Ферма.

Теорема 1. Число Ферма $n = F_k$ при $k > 0$ является простым тогда и только тогда, когда $3^{(n-1)/2} \equiv -1 \bmod n$.

Определение 2. Пусть p – простое число. Числа вида $M_p = 2^p - 1$ называются числами Мерсенна.

Кстати, все четные совершенные числа имеют вид $2^{p-1}M_p$, где M_p является числом Мерсенна. Напомним, совершенным числом называется число, которое равно сумме всех своих делителей, меньших, чем оно само. Например, $28 = 1 + 2 + 4 + 7 + 14$.

Числа Мерсенна редки. В 2001 году было найдено тридцать девятое число Мерсенна для $p = 1\,3466\,917$. Для проверки простоты чисел Мерсенна применяется следующая теорема [7].

Теорема 2. Пусть n – простое число, $n > 2$, $M_n = 2^n - 1$. Рассмотрим последовательность $L_0, L_1, \ldots$, которая определяется соотношениями

$$L_0 = 4, L^2_{j+1} = L^2_j - 2 \bmod n, 0 \le j < n.$$

Число M_n – простое тогда и только тогда, когда $L_{q-2} \equiv 0 \bmod n$.

ПРОВЕРКА НА ПРОСТОТУ

Простые числа необходимы для большинства криптографических систем с открытыми ключами. Теоретический материал к вопросу построения больших простых чисел можно найти в [7] и [11]. Здесь будут сформулированы только некоторые практические подходы к формированию больших простых чисел. Для генерации больших простых чисел могут быть использованы следующие два подхода:

- формируются случайные числа заданного порядка, и при помощи существующих тестов проверяется являются ли они простыми.
- по определенному алгоритму генерируются простые числа, и при помощи определенных тестов производится проверка чисел на простоту.

Сначала рассмотрим те тесты, которые используются при реализации первого подхода формирования простого числа.

Пробное деление

Один из самых простых способов проверки числа p на простоту состоит в последовательном делении числа p на все нечетные числа, которые содержатся в интервале $[2, \sqrt{p}]$. Если в процессе деления получим целый результат, то число p – составное. Если же при переборе всех нечетных чисел из интервала $[2, \sqrt{p}]$ разделить число p на эти числа нацело нельзя, то число p – простое. Данный метод называется *пробным делением*. Этот метод трудоемок по числу арифметических операций, и он используется в основном для проверки небольших простых чисел.

Решето Эратосфена

Если мы хотим составить таблицу всех простых чисел среди чисел 2, 3,…, N, то надо последовательно вычеркнуть все числа, которые делятся

- на 2, кроме 2;
- на 3, кроме 3;
- на 5 кроме 5;
- на следующее число, которое не вычеркнуто, кроме этого числа; и т. д.

В итоге среди чисел от 1 до N останутся лишь простые числа. Для реализации метода нужен большой объем памяти ЭВМ, однако для составления таблиц простых чисел он является наилучшим [11]. Более того, разрабатываются специальные процессоры, на которых операции «просеивания» выполняются очень эффективно [7].

Замечание. Пробное деление и решето Эратосфена можно применять при решении задачи разложения целого числа на множители.

Тест на основе малой теоремы Ферма

Малая теорема Ферма [7] утверждает, что если n простое число, то выполняется условие: при всех $a \in \{2, 3, \ldots, n-1\}$ имеет место сравнение

$$A^{n-1} \equiv 1 \bmod n.$$

На основании этой теоремы можно построить вероятностный алгоритм проверки на простоту числа n. Если для некоторого целого a из интервала

[2, n] соотношение $a^{n-1} \equiv 1 \bmod n$ не выполняется, то число n – составное. Если же теорема выполняется, то вывод, что число n простое, сделать нельзя, так как теорема дает лишь необходимое условие. Поэтому, если для некоторого a имеет место соотношение $a^{n-1} \equiv 1 \bmod n$, то говорят, что число n является псевдопростым по основанию a [1]. Существует бесконечно много пар чисел a и n, где n – составное и псевдопростое. Вообще для любого $a > 1$ существуют бесконечно много псевдопростых чисел по основанию a [1]. Вообще, справедливы следующие два утверждения.

- Если пара (2, n) удовлетворяет сравнению $a^{n-1} \equiv 1 \bmod n$, то и пара чисел (2, $2^n - 1$) также ему удовлетворяет.
- Для любого простого числа n и любого $a > 2$ такого, что $(a^2 - 1, n) = 1$, число $(a^{2n} - 1)/(a^2 - 1)$ является псевдопростым по основанию a.

Определение. Составные числа n, для которых при всех основаниях выполняется сравнение $a^{n-1} \equiv 1 \bmod n$, называются числами Кармайкла.

Схема алгоритма на базе малой теоремы Ферма

Дано число n и параметр $\gamma = 1$ для идентификации результата проверки.

1) делается случайный выбор целого числа a из интервала [2, n];

2) используя алгоритм Эвклида, вычисляется НОД для чисел a и n;

3) если НОД больше 1, то выполняется шаг 7;

4) для числа a проверяется сравнение $a^{n-1} \equiv 1 \bmod n$;

5) если сравнение не выполняется, то определяется параметр $\gamma = 0$ (число составное) переход на шаг 7;

6) если сравнение выполнено, то можно повторить тест;

7) выдать результат проверки ($\gamma = 0$ – число составное).

При завершении работы алгоритма возможны следующие выводы:

- число n – составное ($\gamma = 0$);
- если $\gamma = 1$, то число n является либо простым, либо составным и числом Кармайкла.

Здесь уместно заметить [7], что числа Кармайкла достаточно редки. Так существуют всего 2 163 чисел Кармайкла, которые не превосходят 25 000 000 000, и всего 16 чисел, которые не превосходят числа 100 000.

Тест Соловея – Штрассена

Тест Соловея – Штрассена проверки на простоту числа p базируется на теореме 4.

Теорема 4. Для любого нечетного n следующие условия эквивалентны:

- n – простое;
- для любого $a \in Z_n^*$ выполняется сравнение $a^{(n-1)/2} \bmod n \equiv L(a, n)$, где Z_n^* – мультипликативная группа, элементами которой являются элементы $a \in Z_n$ (Z_n – кольцо вычетов по модулю n).

Для проверки простоты числа p используется алгоритм вычисления символа Якоби.

Схема алгоритма на базе малой теоремы Ферма

Пусть дано нечетное число p. Надо проверить является ли число p простым.

1. Выбирается случайное число a, меньшее p.
2. Если НОД $(a, p) \neq 1$, то p – составное число.
3. Вычисляется сравнение $L(a, p) \equiv a^{(p-1)/2} \bmod p$.
4. Вычисляется символ Якоби $J(a, p)$.
5. Если $L(a, p) \neq J(a, p)$, то число p – составное.
6. Если $L(a, p) = J(a, p)$, то вероятность того, что число p – составное не превышает 50 %.

Если проверка повторяется k раз, то вероятность того, что число p – составное, не превышает $1/2^k$.

Тест Рабина – Миллера

Обоснование алгоритма Рабина – Миллера можно найти в [3]. Здесь дадим только самые общие соображения. Известно, что если число p – простое, то уравнение $x^2 \equiv 1 \bmod p$ имеет лишь два решения: $x \equiv \pm 1 \bmod p$. Итак, пусть p – нечетное целое число, которое надо проверить на простоту. Если p – простое, то по теореме Ферма для любого целого a взаимно простого с p выполняется сравнение $a^{m-1} \equiv 1 \bmod p$. Так как $p - 1$ – четно, то получаем $a^{(m-1)/2} \equiv \pm 1 \bmod p$. Если оказывается, что $(m - 1)/2$ – четно, то можно повторить рассуждение, при котором получим, что $a^{(m-1)/4} \equiv \pm 1 \bmod p$, и т. д.

Поэтому, чтобы проверить на простоту нечетное число p, выбираем случайным образом число a из интервала $[1, p - 1]$ и вычисляем

$$a^t \bmod p,\ a^{2t} \bmod p,\ \ldots,\ a^{\beta t} \bmod p,$$

где t – нечетное и число $\beta = 2^s$. Если одно из этих чисел не совпадает с +1 или −1, то можно сделать вывод, что число p является числом составным.

Если значения чисел совпадают с +1 или –1, то повторяем этот тест k раз. После повторения этого теста k раз вероятность того, что составное число p не будет выявлено, не превосходит $1/4^k$. После высказанных соображений перейдем к формулировке алгоритма Рабина – Миллера. В некоторой литературе данный алгоритм называют тестом Миллера – Рабина [4].

Схема алгоритма Рабина – Миллера

Пусть дано нечетное число p. Надо проверить является ли число p простым. Далее предположим, что $p - 1 = 2^s t$.

1. Выбираем случайное число a, меньшее p и определяем $k = 0$.

2. Вычисляем с помощью алгоритма Эвклида НОД двух чисел a и p. Если НОД $(a, p) \neq 1$, то p – составное число.

3. Вычисляем $b \equiv a^t \bmod p$. Если $b = 1$ или $b = p - 1$, то число p вероятно простое.

4. Если $b \neq 1$ и $b \neq p - 1$, то вычисляем $b \equiv b^2 \bmod p$ и $k = k + 1$.

5. Если число $b = p - 1$, то число p вероятно простое. Перейти на шаг 7.

6. Пока $k < s$ выполнять пункт 4.

7. Завершить работу алгоритма.

Рассмотрим примеры.

Пример. Пусть $p = 181$. Имеем $p - 1 = 45 \times 2^2$. По представленному разложению определяем значение параметра $t = 45$.

1. Выбираем случайное число $a = 52 < p$ и определяем $k = 0$.

2. Используем алгоритм Эвклида для вычисления НОД двух чисел 52 и 181:

делим число 181 на число 52, получаем $181 = 52 \cdot 3 + 25$;

делим число 52 на число 25, получаем $52 = 25 \cdot 2 + 2$;

делим число 25 на число 2, получаем $25 = 12 \cdot 2 + 1$;

делим число 2 на число 1, получаем $2 = 1 \cdot 2 + 0$;

получаем, что НОД двух чисел 181 и 52 равен 1.

Так как НОД не позволяет установить является ли число 181 составным, то продолжаем выполнять алгоритм Рабина – Миллера.

3. Последовательно вычисляем

$$b \equiv 52^t \bmod 181 \equiv 52^{45} \bmod 181:$$

$$52^2 \bmod 181 \equiv 2704 \bmod 181 \equiv 170 \bmod 181,$$

$$52^4 \bmod 181 \equiv 170^2 \bmod 181 \equiv 28900 \bmod 181 \equiv 121 \bmod 181,$$

$$52^8 \bmod 181 \equiv 121^2 \bmod 181 \equiv 14641 \bmod 181 \equiv 161 \bmod 181,$$

$$52^{16} \bmod 181 \equiv 161^2 \bmod 181 \equiv 25921 \bmod 181 \equiv 38 \bmod 181,$$

$$52^{32} \bmod 181 \equiv 38^{2`} \bmod 181 \equiv 1444 \bmod 181 \equiv 177 \bmod 181,$$

$$52^{40} \bmod 181 \equiv (52^{32} \bmod 181)(52^{8} \bmod 181) \equiv$$
$$\equiv (177 \times 161) \bmod 181 \equiv 28497 \bmod 181 \equiv 80 \bmod 181,$$

$$52^{41} \bmod 181 \equiv (52^{40} \bmod 181)(52 \bmod 181) \equiv (80 \times 52) \bmod 181 \equiv$$
$$\equiv 4160 \bmod 181 \equiv 178 \bmod 181,$$

$$52^{45} \bmod 181 \equiv (52^{41} \bmod 181)(52^{4} \bmod 181) \equiv (178 \times 121) \bmod 181 \equiv$$
$$\equiv 21538 \bmod 181 = 180 \bmod 181.$$

Итак, получили $b = 180 = p - 1$. Откуда следует, что число $p = 181$ вероятно простое.

Замечание. Для генерации даже небольших простых чисел вычисления довольно громоздки. Поэтому для реальных чисел, несомненно, подобные проверки чисел на простоту надо делать при помощи программ на компьютере.

В [10] дается некоторое руководство при генерации простых чисел для практических приложений. Это руководство сводится к реализации нескольких этапов (шагов).

1. Сгенерируйте случайное n-битовое число p.

2. Установите старший и младший биты равными 1. Старший бит в этом случае гарантирует требуемую длину простого числа, а младший – его нечетность.

3. Убедитесь, что число p не делится на малые простые числа 3, 5, 7, 11 и т. д. Наиболее надежна проверка делимости на все простые числа, меньше 2000.

4. Выполните тест Рабина – Миллера для некоторого случайного числа a. Если p проходит тест, то сгенерируйте другое случайное число a и повторите тест. Для практических приложений достаточно повторить тест Рабина – Миллера пять раз.

5. Если p не проходит один из тестов, надо сгенерировать другое число p и повторить данное руководство снова.

Другое число p, если оно оказалось не простым, можно получить не генерируя новое, а последовательно перебирая все целые, начиная от $p + 1$, $p + 2$, и т. д., пока не найдется простое число.

Перейдем теперь к вопросу о генерировании больших простых чисел.

Построение больших простых чисел и детерминированные алгоритмы проверки чисел на простоту

Рассмотрим еще один способ формирования простых чисел. Этот способ базируется на определенной процедуре генерирования простых чисел, проверка которых осуществляется с помощью тестов на простоту. Такой подход применяется, например, в стандарте электронной цифровой подписи (ЭЦП) ГОСТ Р 34.10–94 и основывается на следующей теореме [12].

Теорема 5. Пусть $p = qN + 1$, где q – нечетное простое число, N – четное число и $p < (2q + 1)^2$. Число p является простым, если выполняются два условия:

1) $2^{qN} = 1 \bmod p$;

2) $2^N \neq 1 \bmod p$.

Генерация простого числа с использованием теоремы 5 осуществляется по несколько упрощенной в принятом стандарте схеме [12]. Пусть требуется сформировать простое число p длины $t \geq 17$ бит. С этой целью строится убывающая последовательность чисел $\{t_i\}$, где $i = 0, 1, \ldots, s$, для которых $t_0 = t$, $t_i = [t_{i-1}/2]$. Далее последовательно вырабатывается последовательность простых чисел $p_s, p_{s-1}, \ldots p_0$ для всех $i = 1, \ldots, s$. Генерация простого числа p_{i-1} осуществляется с использованием формулы $p_{i-1} = p_i N + 1$, где число N удовлетворяет таким условиям [12]:

- N – четное;
- N – такое, что длина числа $p_{i-1} = p_i N + 1$ в точности должна быть равна t_{i-1}.

В стандарте ГОСТ дается некоторый алгоритм вычисления числа N. Для учебного варианта N – случайное четное число, которое получают с помощью датчика случайных чисел (если N – нечетно, то $N = N + 1$).

Число p_{i-1} считается полученным, если одновременно выполнены следующие два условия:

1) $2^{\beta} = 1 \bmod p_i$, $\beta = p_{i-1} N$;

2) $2^N \neq 1 \bmod p_i$.

Если хотя бы одно из условий не выполняется, то значение числа N увеличивается на 2, и вычисляется новое значение p_{i-1}, которое снова проверяется на простоту по двум условиям. Данный процесс продолжается до тех пор, пока не будет получено простое число p_{i-1} (т. е. условия 1 и 2 алгоритма генерации простого числа будут выполнены).

Заметим, что с 2002 года вышеупомянутый отечественный стандарт ЭЦП заменен на новый ГОСТ Р 34.10–2001 [7].

Проверка чисел Мерсенна на простоту

Напомним, что число $M_s = 2^s - 1$ ($s \geq 2$ – простое число) называется числом Мерсенна. Критерием простоты чисел Мерсенна служит следующее утверждение [3].

Теорема. Число Мерсенна $M_s = 2^s - 1$, где $s \geq 3$ – нечетное число, является простым тогда и только тогда, когда

– число s – простое;

– выполняется сравнение $L_{n-2} \equiv 0 \bmod M_s$, где последовательность $\{L_k\}$ формируется по такому правилу:

$$L_0 = 4, L_{k+1} = (L^2_k - 2) \bmod M_s \text{ при } k \geq 0.$$

РАЗЛОЖЕНИЕ ЧИСЕЛ НА МНОЖИТЕЛИ

Задача разложения составного числа на множители – важная задача в криптографии. Эта задача более сложная, чем задача проверки чисел на простоту. Естественно, прежде чем раскладывать число на множители, надо убедиться, что исследуемое число не является простым. Ниже, не ограничивая себя в общности, будем искать разложение числа на два сомножителя, т. е. предположим, что исследуемое число n представимо в виде $n = p \cdot q$, где для целых чисел p и q выполняются неравенства $1 < p < n$, $1 < q < n$.

Метод пробного деления

Напомним, что пробное деление и решето Эратосфена можно применять при решении задачи разложения целого числа на множители. То есть если n – составное число, то, перебирая простые числа 3, 5, 7, ..., проверяем, являются ли эти числа делителями числа n. Процесс поиска можно немного ускорить, если заранее вычислить произведения простых чисел $s = 1 \cdot 2 \cdot 3\ldots$ (так, чтобы данное произведение было меньше $\sqrt{n}$) и (используя алгоритм Евклида) определить общий делитель $d = \text{НОД}(n,s)$.

Пример. Разложим на множители число n = 84 257 901 методом пробного деления [3].

Составим базу данных из произведений простых чисел по три:

$$q_1 = 3 \cdot 5 \cdot 7 = 105;\ q_2 = 11 \cdot 13 \cdot 17 = 2\,431;$$

$$q_3 = 19 \cdot 23 \cdot 29 = 12\,673.$$

Вычисляем

$$d_{11} = \text{НОД}(n\ , q_1) = \text{НОД}(84\,257\,901, 105) = 21,\ n = n/d_{11} = 4\,012\,281;$$

$$d_{12} = \text{НОД}(n, q_1) = \text{НОД}(4\,012\,281, 105) = 21,\ n = n/d_{12} = 191\,061;$$

$$d_{13} = \text{НОД}(n, q_1) = \text{НОД}(191\,061, 105) = 3,\ n = n/d_{13} = 63\,687;$$

$$d_{14} = \text{НОД}(n, q_1) = \text{НОД}(63\,687, 105) = 3,\ n = n/d_{14} = 21\,229.$$

Значит, 84 257 901 делится на 3^4 и на 7^2 аналогично,

$$d_{21} = \text{НОД}(n, q_2) = \text{НОД}(21\,229, 2\,431) = 13,\ n = n/d_{21} = 1\,633;$$

$$d_{31} = \text{НОД}(n, q_3) = \text{НОД}(1\,633, 12\,673) = 23,\ n = n/d_{31} = 71.$$

Число 71 – простое. Таким образом, получаем

$$84\,257\,901 = 3^4 \cdot 7^2 \cdot 13 \cdot 23 \cdot 71.$$

Факторизация Ферма

Существует способ факторизации составного числа n, который эффективен, если n – произведение двух целых чисел, близких друг к другу [14]. Этот метод разложения составного числа n на множители называется *факторизацией Ферма*. Он основан на теореме о представлении числа в виде разности двух квадратов.

Теорема. Если $n > 1$ нечетно, то существует взаимно однозначное соответствие между разложениями на множители $n = p \cdot q$, $q \geq p > 0$ и представлениями в виде разности квадратов $n = x^2 - y^2$, $x > y > 0$. Это соответствие имеет вид $x = (q + p)/2$, $y = (q - p)/2$, $q = x + y$, $p = x - y$.

Доказательство. Пусть $n = pq$, где $q \geq p$. Полагая $x = (q + p) / 2$, $y = (q - p)/2$, получаем нужную пару $\{x, y\}$.

Обратно, пусть $n = x^2 - y^2 = (x + y)(x - y)$. Полагая $q = x + y$, $p = x - y$, получаем требуемое разложение.

Метод Ферма заключается в том, что можно найти пару (x, y), перебирая в качестве кандидатов на значение x числа $\lceil\sqrt{n}\rceil + 1$, $\lceil\sqrt{n}\rceil + 2$, … и проверяя для каждого из них равенства $\left(\lceil\sqrt{n}\rceil + i\right)^2 - n = y^2$.

Для отбраковки ложных значений x можно воспользоваться тем, что если число не является квадратом, то оно с большой вероятностью не будет и квадратичным вычетом для одного из небольших простых чисел p. Последнее свойство легко проверяется путем вычисления соответствующего символа Лежандра.

Алгоритм. Факторизация Ферма [7].

Вход: n – нечетное число, $p_1, \ldots, p_k$ – небольшие простые числа.

Шаг 1. Проверить $p_i \mid n$, $i = 1, \ldots, k$. Если да, то делитель найден.

Шаг 2. Для каждого x от $\lceil\sqrt{n}\rceil + 1$ до $\lceil\sqrt{n}\rceil + n_0$ вычислить величины

$$t = x^2 - n,\ t_i = t \bmod p_i,\ i = 1, \ldots, k.$$

Шаг 3. Если хотя бы для одного $i = 1, \ldots, k$ выполнено одно из условий:

- $t_i = 0$ и p^2 не делит t или
- $t_i \neq 0$ и $L(t_i, p_i) = -1$ ($L(t_i, p_i)$ – символ Лежандра), то нужно перейти к следующему x на шаге 2.

В противном случае перейти к шагу 4.

Шаг 4. Проверить, является ли $t = x^2 - n$ полным квадратом.

Если $x^2 - n = y^2$, то выдать ответ: «n – составное, $n = a \cdot b$, $a = x + y$, $b = x - y$».

Если $t = x^2 - n$ – не полный квадрат, то перейти к следующему x на шаге 2.

По сути данный алгоритм, подобно методу «пробных делений», является разновидностью метода перебора всех возможных делителей. Однако в отличие от метода «пробных делений», с помощью которого находится наименьший делитель, данный метод позволяет находить наибольший делитель числа n, не превосходящий $\sqrt{n}$.

Пример 1. У числа 15 два делителя, не меньших, чем $\sqrt{15}$, – это числа 5 и 15. Тогда получаем два представления: $15 = pq = 3 \cdot 5$, откуда $s = 4$, $t = 1$ и $15 = 4^2 - 1^2$; $15 = pq = 1 \cdot 15$, откуда $s = 8$, $t = 7$ и $15 = 8^2 - 7^2$.

Пример 2. У числа 9 два делителя не меньших, чем $\sqrt{9}$, – числа 3 и 9. Тогда получаем два представления:

$$9 = pq = 3 \cdot 3, \text{ откуда } s = 3,\ t = 0 \text{ и } 9 = 3^2 - 2^2;$$

$$9 = pq = 1 \cdot 9, \text{ откуда } s = 5,\ t = 4 \text{ и } 9 = 5^2 - 4^2;$$

Пример 3. У числа 11 один делитель, не меньший, чем $\sqrt{11}$, – само число 11. Тогда получаем представление $11 = pq = 1 \cdot 11$, откуда $s = 6$, $t = 5$ и $11 = 6^2 - 5^2$.

Понятно, что чем больше разность между числами p и q, тем более трудоемким становится метод Ферма. В этом случае можно воспользоваться обобщенным методом Ферма [3]: для небольшого целого числа k последовательно вычислять $s = \left\lceil \sqrt{kn} \right\rceil + 1$, $s = \left\lceil \sqrt{kn} \right\rceil + 2$, ..., пока не получится такое число s, что разность $s^2 - kn$ является полным квадратом, т. е. равна r^2. Отсюда $(s + t)(s - t) = kn$, и значит, числа $s - t$ и n имеют нетривиальный общий делитель.

Пример 4. Разложим на множители число $n = 5\ 338\ 771$. При разложении методом Ферма лишь на 160-м шаге получим

$$s = \left\lceil \sqrt{n} \right\rceil + 160 = 2\ 310 + 160 = 2\ 470,$$

откуда

$$r^2 = 2\ 4702^2 - 5\ 338\ 771 = 762\ 129,$$

$$t = 873,\ s + t = 3\ 343,\ s - t = 1\ 597.$$

Используя обобщенный метод Ферма, выбрав $k = 8$, уже со второй попытки получим [3]

$$\left(\left\lceil \sqrt{kn} \right\rceil + 2\right)^2 - kn = \left(\left[\sqrt{\alpha} \right] + 2 \right)^2 - 8 \cdot 5\ 338\ 771 =$$
$$= 6537^2 - 8 \cdot 5\ 338\ 771 = 22\ 201 = 149^2,$$

где $\alpha = 8 \cdot 5\ 338\ 771$. Таким образом,

$$s = 6537,\ t = 149,\ s - t = 6537 - 149 = 6388,$$

$$\text{НОД}(6388, n) = 1597.$$

Действительно, $n = 1597 \cdot 3343$.

На практике подобрать число k в обобщенном методе Ферма не всегда легко, поэтому для разложения целого числа n достаточно найти такие целые числа x, y, что $x^2 \equiv y^2 \pmod{n}$. Для поиска таких чисел в пособиях [3] и [7] формулируется алгоритм, построенный на методе Диксона. Этот метод использует понятие факторной базы. Факторной базой называется некоторое множество $B = \{p_0, p_1..., p_h,\}$ небольших простых чисел. Обычно p_i ($i = 1, 2, \ldots, h$) – это простые числа, которые не превосходят некоторого числа M, а h – число таких простых чисел.

ρ-метод Полларда

Пусть n – нечетное составное число, $S = \{0, 1, ..., n - 1\}$ и f: $S \to S$ – случайное отображение, обладающее сжимающими свойствами, например $f(x) \equiv x^2 + 1 \pmod{n}$. Основная идея метода: выбираем случайный элемент $x_0 \in S$ и строим последовательность $x_0, x_1, x_2, \ldots$, определяемую рекуррентным соотношением $x_{i+1} = f(x_i)$, где $i \geq 0$, до тех пор, пока не найдем такие числа i, j, что $i < j$ и $x_i = x_j$. Поскольку множество S конечно, такие индексы i, j существуют (последовательность «зацикливается»). Последовательность $\{ x_i,\}$ будет состоять из «хвоста» $x_0, x_1, x_2, \ldots, x_{i-1}$ длины $O(\sqrt{\pi \cdot n/8})$ и цикла $x_i = x_j, x_{i+1}, \ldots, x_{j-1}$ той же длины [2].

Если p – простой делитель числа n и $x_i \equiv x_j \pmod{p}$, то разность $x_i - x_j$ делится на p и НОД$(x_i - x_j, n) > 1$. Число p нам нужно найти, поэтому все вычисления в алгоритме будем проводить по модулю n и на каждом шаге вычислять $d =$ НОД $x_i - x_j, n$). Нетривиальный наибольший общий делитель $1 < d < n$ (когда числа x_i и x_j принадлежат одному классу вычетов по модулю p, но разным классам вычетов по модулю n) как раз и будет искомым делителем p числа n . Случай $d = n$ имеет место с пренебрежимо малой вероятностью.

Если последовательно вычислять значения $x_{i+1} = f(x_i)$, запоминать их и искать равные, то число шагов алгоритма в среднем равно $\sqrt{\pi \cdot n/2}$. Кроме того, временная и емкостная сложности равны $O(\sqrt{\pi})$.

Необходимость хранения большого количества информации устраняется, если применить алгоритм Флойда [10]: начиная с $i = 1$, рекурсивно вычислять (x_i, x_{2i}) из (x_{i-1}, x_{2i-2}), пока не найдется такое число k, что $x_m = x_{2m}$.

Обозначим длину «хвоста» последовательности через λ, длину цикла через μ и оценим максимальное число шагов m до входа в цикл. Понятно, что $x_k = x_i$ при $k \geq i$ тогда и только тогда, когда разность $k - i$ делится на μ (от x_i до x_k необходимо пройти хотя бы один цикл). Кроме того, должно

выполняться неравенство $i \geq \lambda$, поскольку два значения из «хвоста» совпасть не могут – только из цикла. При использовании алгоритма Флойда получаем $x_{2i} = x_i$ тогда и только тогда, когда i делится на μ и $i \geq \lambda$.

Найдем верхнюю оценку для минимального значения i_{min}, при котором выполняется равенство $x_{2i} = x_i$. Перенумеруем все элементы цикла последовательности $\{x_i\}$ числами от 0 до $\mu - 1$. После λ шагов расстояние между точками x_λ и $x_{2\lambda}$ будет равно $l \equiv \lambda \pmod{\mu}$: по «одиночной» траектории $\{x_{2i}\}$ сделали λ шагов, по «двойной» траектории $\{x_{2i}\}$ – 2λ шагов. На каждом шаге расстояние между точками «одиночной» и «двойной» траектории увеличивается на 1: после первого шага оно равно $l + 1$, после второго шага $l + 2$, ..., после произвольного числа s шагов – $l + s$. Для совпадения точек двух траекторий необходимо, чтобы $l + s = \mu$, т. е. $s = \mu - l = \mu - (\lambda \pmod{\mu})$, где вычитаемое – наименьший неотрицательный вычет по модулю μ. Учитывая, что предварительно до цикла нужно еще пройти весь «хвост» длины λ, получаем следующие границы: $\lambda < i_{\min} < \lambda + \mu - \lambda \pmod{\mu}$.

Замечание. Графически последовательность $\{s_i\}$ можно изобразить так, что ее «хвост» и цикл будут образовывать греческую букву «ро» Отсюда и название метода.

Алгоритм 1. ρ-метод Полларда.

Вход. Число n, начальное значение c, функция f, обладающая сжимающими свойствами.

Выход. Нетривиальный делитель p числа n.

1. Положить $a \leftarrow c$, $b \leftarrow c$.

2. Вычислить $a \leftarrow f(a) \pmod{n}$, $b \leftarrow f(b) \pmod{n}$, $b \leftarrow f(b) \pmod{n}$.

3. Найти $d \leftarrow$ НОД$(a - b, n)$.

4. Если $1 < d < n$, то положить $p \leftarrow d$ и результат: p. При $d = n$ результат: «Делитель не найден»; при $d = 1$ вернуться на шаг 2.

Пример. Найдем p-методом Полларда нетривиальный делитель числа $n = 1\,359\,331$. Положим $c = 1$ и $f(x) \equiv x^2 + 5 \pmod{n}$.

Работа алгоритма иллюстрируется следующей таблицей:

i	A	B	d	НОД$(a - b, n)$
1	16	141		1
2	41	123 939		1
3	1 686	391 594		1
4	123 939	438 157		1
5	435 426	582 738		1
6	391 594	1 144 026		1
7	1 090 062	885 749		1 181

Таким образом, 1 181 является нетривиальным делителем числа 1 359 331 (действительно, $1\,359\,331 = 1\,181 \cdot 1\,151$).

Пример. Найдем нетривиальный делитель числа n = 1 373 503. Положим $c = 2$ и $f(x) \equiv x^2 - 2 \pmod{n}$. $a = b = 2$ (неподвижная точка) и d = 1 373 809. Изображение $f(x) \equiv x^2 - 22 \pmod{n}$, за пять итераций получаем a = 1 163, b = 742 847 и d = 1 181 (действительно, 1 373 503 = 1 181 · 1 163.

В предположении, что функция f ведет себя как случайная, алгоритм 6.1 требует для нахождения делителя p числа n порядка $O(\sqrt{n})$ модульных умножений. Следовательно, для нахождения нетривиального делителя числа n требуется $O(\sqrt[4]{n})$ модульных умножений.

(p–1)-метод Полларда

Пусть n – нечетное составное число и p – его нетривиальный делитель. *(p–1)*-метод особенно эффективен при разложении таких чисел n, для которых число $p - 1$ сильно составное [10; 11].

Определение 1. Пусть $B = \{p_1, p_2, \ldots, p_s\}$ – множество различных простых чисел. Назовем множество B **базой разложения.** Целое число назовем ***B*-гладким**, если все его простые делители являются элементами множества B.

Идея метода такова. Пусть $n = pq$ и пусть каноническое разложение числа $p - 1$ имеет вид $p - 1 = p_1^{\alpha_1} p_2^{\alpha_1} \ldots p_s^{\alpha_s}$. Найдем максимальные показатели $l_1, l_2, \ldots, 1_s$ для которых $p_i^{l_i} \le n$. Прологарифмируем обе части этого неравенства: $l_i \ln p_i \le \ln n$, откуда имеем

$$l_i \le \left[\frac{\ln n}{\ln p_i} \right].$$

Вычислим

$$M = p_1 \left[\frac{\ln n}{\ln p_1} \right] p_2 \left[\frac{\ln n}{\ln p_2} \right] \ldots p_s \left[\frac{\ln n}{\ln p_s} \right],$$

тогда $M = (p-1) \cdot z$ для некоторого целого числа z.

Согласно малой теореме Ферма выполняется сравнение $a^{p-1} \pmod{p}$ для любого целого a, взаимно простого с p. Возводя обе части этого сравнения в степень z, получаем $a^M \equiv 1 \pmod{p}$.

Обозначим d = НОД($a^M - 1$, n). Если $a^M \equiv 1 \pmod{n}$, то число d должно делиться на p, поскольку разность $a^M - 1$ делится на p и число n делится на p.

Алгоритм 1. (p–1)-метод Полларда.

Вход. Составное число n .

Выход. Нетривиальный делитель p числа n.

1. Выбрать базу разложения $B = \{pi, pi, ..., ps\}$.

2. Выбрать случайное целое a, $2 \le a < n - 2$, и вычислить $d \leftarrow$ НОД(a, n). При $d \ge 2$ положить $p \leftarrow d$ и результат: p.

3. Для $i = 1,2, ..., n$ выполнить следующие действия.

3.1. Вычислить

$$l \leftarrow \left[\frac{\ln n}{\ln p_i} \right];$$

3.2. Положить $a \leftarrow a^{p_i^l} \bmod n$.

4. Вычислить $d \leftarrow$ НОД($a - 1$, n).

5. При $d = 1$ или $d = n$ результат: «Делитель не найден». В противном случае положить $p \leftarrow d$ результат: p.

Равенство $d = n$ на шаге 5 означает, что каноническое разложение числа $q - 1$ включает в себя те же простые числа, что и разложение числа $p - 1$. В силу малой теоремы Ферма $a^{p-1} \equiv 1 \pmod p$ и $a^{q-1} \equiv 1 \pmod q$, а поскольку в этом случае число M делится не только на $p - 1$, но и на $q - 1$, получаем

$$a^M \equiv 1 \pmod p \text{ и } a^M \equiv 1 \pmod q \text{ тогда } a^M \equiv 1 \pmod{pq}.$$

Пример. Разложим (p–1)-методом число n = 1 728 239. Выбираем базу разложения из первых трех простых чисел: $B = \{2, 3, 5\}$. Полагаем $a = 2$ – число, очевидно, взаимно простое с нечетным n.

Вычисляем последовательно:

$$l = \left[\frac{\ln n}{\ln 2} \right] = 20, a \equiv a^{2^{20}} \equiv 1357228 \left(\bmod n \right);$$

$$l = \left[\frac{\ln n}{\ln 3} \right] = 13; a \equiv a^{3^{13}} \equiv 987665 \left(\bmod n \right);$$

$$l = \left[\frac{\ln n}{\ln 5} \right] = 8, a \equiv a^{5^8} \equiv 74463 \pmod n \cdot$$

Находим d = НОД($a - 1$, n) = НОД(74 462, 1 728 239) = 1201.

Таким образом, p = 1 201 – нетривиальный делитель числа 1 728 239. Действительно, 1 728 239 = 1201 · 1439. При этом $p - 1 = 1200 = 2^4 \cdot 3 \cdot 5^2$. Заметим, что число 1438 = 1439 – 1 не является B-гладким: 1438 = 2 · 719.

Пример. Пусть n = 1 557 697. Выбираем ту же базу разложения, что и в предыдущем случае, и снова полагаем a = 2.

Вычисляем последовательно:

$$l = \left[\frac{\ln n}{\ln 2}\right] = 20, a \equiv a^{2^{20}} \equiv 301549 \left(\mathrm{mod}\, n\right);$$

$$l = \left[\frac{\ln n}{\ln 3}\right] = 13; a \equiv a^{3^{13}} \equiv 953296 \left(\mathrm{mod}\, n\right);$$

$$l = \left[\frac{\ln n}{\ln 5}\right] = 8; a \equiv a^{3^{8}} \equiv 1 \left(\mathrm{mod}\, n\right).$$

Находим d = НОД($a - 1$, n) = НОД(0, 1 557 697) = 1 557 697. Таким образом, $d = n$, и нетривиальный делитель числа n не найден. Заметим, что здесь 1 557 697 = 1201 · 1297, при этом $1297 - 1 = 2^4 \cdot 3^4$, т. е. число 1296 тоже является B-гладким.

Пример. Пусть n = 3 865 489. Если строить базу разложения, начиная с малых простых чисел 2, 3, 5, ..., то для нахождения нетривиального делителя при случайных значениях a, взаимно простых с n, потребуется база, содержащая не менее 162 первых простых чисел. Это объясняется тем, что $n = pq$ = 1907 · 2027, где $1907 - 1 = 2 \cdot 953$ (953 – это 162-е простое число) и $2027 - 1 = 2 \cdot 1013$. Если же $a \equiv 1 \pmod{p}$ (или $a \equiv 1 \bmod q$), то на шаге 4 получаем d = НОД($a - 1$, n) = p (или d = НОД($a - 1$,n) = q).

В литературе, например, [3], [11–14] можно ознакомиться с некоторыми другими алгоритмами факторизации целого числа, которые не представлены в данном пособии.

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ НАД БОЛЬШИМИ ЧИСЛАМИ

Диапазон чисел, которые используются в реальных задачах криптографии, доходит до нескольких сот и даже тысячи десятичных цифр. Такой диапазон чисел не соответствует базовым типам данных современных компьютеров. Число, которое состоит из несколько сот (и даже тысяч) десятичных знаков, нельзя записать как единый объект ни в одно базовое устройство компьютера. Поэтому компьютерное представление таких чисел и операции над ними приходится реализовывать самостоятельно в виде некоторых программ.

Представление целого числа a в системе счисления с основанием B имеет вид

$$a = a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_1B + a_0,$$

где $0 \le a_i < B$. Заметим, что данное представление целого числа аналогично представлению полинома степени $n-1$:

$$a(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$$

с коэффициентами a_i, $0 \le a_i < B$.

Сделаем одно замечание для всех последующих алгоритмов, которые определяют значение арифметических операций для больших чисел.

Замечание. Знак числа, как и место десятичной точки, можно запомнить в отдельной переменной и учитывать при выполнении операций. Далее речь пойдет только о целых неотрицательных числах.

Теперь сформулируем основные алгоритмы для выполнения арифметических операций с большими целыми числами (или, иначе, чисел с произвольным количеством разрядов) [3; 12; 13].

Сложение

Рассмотрим два целых числа a, b по некоторому основанию B, т. е.

$$a = a_{n-1}B^{n-1} + a_{n-2}B^{n-2} + \ldots + a_1B + a_0,$$

$$b = b_{n-1}B^{n-1} + b_{n-2}B^{n-2} + \ldots + b_1B + b_0,$$

где $0 \le a_i < B$, $0 \le b_i < B$. Для сложения чисел используется следующий алгоритм.

Алгоритм 1. Найти сумму $w = a + b$ [3; 12; 13].

1. Определить $c = 0$, $i = 0$.
2. Для $i = 0, 1, \ldots, n-1$ выполнить шаги 3–5 алгоритма.
3. Вычислить $t = a_i + b_i + c$.
4. Положить $c = t \bmod B$.
5. $w_i = c$.
6. Положить $w_n = t$.
7. Результат $w = w_n B_n + w_{n-1}B^{n-1} + w_{n-2}B^{n-2} + \ldots + w_1 B + w_0$,

Вычитание

Алгоритм 2. Найти разность $w = a - b$, где $0 < b \le a$ [3; 12; 13].

1. Определить $c = 0$, $i = 0$.
2. Для $i = 0, 1, \ldots, n-1$ выполнить шаги 3–5 алгоритма.
3. Вычислить $t = a_i - b_i + c$.
4. Положить $c = t \bmod B$.
5. $w_i = c$.
6. Результат

$$w = w_{n-1}B^{n-1} + w_{n+m-2}B^{n+m-2} + \ldots + w_1 B + w_0.$$

Заметим, что в процессе выполнения алгоритма, переменная c может принимать отрицательное значение, т. е. может выполняться неравенство $c < 0$.

Умножение

Операции умножения и деления являются наиболее трудоемкими. В настоящее время существует много методов умножения больших чисел. Здесь мы рассмотрим алгоритм, который моделирует школьный метод умножения в столбик [3; 12; 13].

Алгоритм 3. Найти произведение двух чисел a (n – разрядное число) и b(m – разрядное число): $w = a \cdot b$.

1. Для $i = 0, 1, \ldots, n-1$
2. Определить $w_i = 0$.
3. Для $i = 0, 1, \ldots, m-1$ (выполнить шаги 4–9 алгоритма).
4. Определить $c = 0$.
5. Для $j = 0, 1, \ldots, n-1$ (выполнить шаги 6–8 алгоритма).
6. Вычислить $t = w_{i+j} + a_i \cdot b_j + c$.
7. Положить $w_{i+j} = t \bmod B$.
8. $c = t \bmod B$.
9. $w_{i+n} = c$.
10. Результат $w_{n+m-1}B^{n+m-1} + w_{n+m-2}B^{n+m-2} + \ldots + w_1 B + w_0$.

Одним из самых трудных алгоритмов умножения двух чисел является алгоритм, который базируется на быстром преобразовании Фурье [3, 7]. Недостаток этого алгоритма заключается в том, что выигрыш в скорости умножения чисел по сравнению, допустим, с изложенным алгоритмом достигается лишь тогда, когда длина сомножителей 8000–10000 двоичных разрядов [13].

Заметим, что для реальных задач криптографии часто используют числа гораздо большей разрядности.

Деление

В литературе существуют описания различных методы деления чисел (см., например: 3; 11; 12; 13). Здесь мы рассмотрим алгоритм деления «в столбик» [3]. Обоснование алгоритма дано в [3].

Алгоритм 4. Найти результат деления двух чисел a ($n + m$ – разрядное число) и b (n – разрядное число) [3], [13]. Начальные данные алгоритма: целые числа $a = (a_{m+n-1}, \ldots, a_0)_B$, $b = (b_{n-1}, \ldots, b_0)_B$, где B – основание системы счисления.

Результат выполнения алгоритма: целые числа $q = (q_m, q_{m-1}, \ldots, q_0)_B$ (частное от деления двух чисел) и $r = (r_n, r_{n-1}, \ldots, r_0)_B$ (остаток от деления двух чисел), такие, что $a = qb + r$, $0 \le r < b$.

1. Определить такое целое число $d > 0$, что $d \cdot b_{n-1} \ge [B/2]$.
2. Положить $(r_{m+n}, r_{m+n-1}, \ldots, r_0)_B \leftarrow (0, a_{m+n-1}, a_{m+n-2}, \ldots, a_0)_B$.
3. Для $i = m + n, m + n - 1, \ldots, n$ выполнить следующие действия (шаги 4–7).
4. Определить $Q = \min([(R_i \cdot B + R_{i-1})/V_{n-1}], B - 1)$, где $R_i = r_i \cdot d$, $V_{n-1} = b_{n-1} \cdot d$, а выражение в квадратных скобках означает целую часть числа.
5. Пока $V_{n-2} \cdot Q > (R_i B + R_{i-1} - QV_{n-1})B + R_{i-2}$ выполнять $Q = Q - 1$.
6. Вычислить $w = r - bQ$ и, если $w < 0$, положить $Q = Q - 1$.
7. Положить $w = r - bQ$ и $q_{i-n} = Q$.
8. Результат: $q = (q_m, q_{m-1}, \ldots, q_0)_B$, $r = (r_n, r_{n-1}, \ldots, r_0)_B$.

Модифицированное деление столбиком

Входные данные: большое неотрицательное целое $n + 1$ – разрядное число x и большое неотрицательное целое $t + 1$ – разрядное число y по основанию b, причем $n \ge t \ge 1$ и $y_t \ne 0$.

Выходные данные: частное $q = (q_{n-t} q_{n-t-1} \ldots q_0)_b$ и остаток от деления $r = (r_t r_{t-1} \ldots r_0)_b$ по основанию b такие, что $x = q \cdot y + r$, причем $0 \le r < y$.

1. Цикл по j от 0 до $(n - t)$: $q_j \leftarrow 0$.
2. Пока $(x \ge y \cdot b^{n-t})$: $q_{n-t} \leftarrow q_{n-t} + 1$ и $x = x - y \cdot b^{n-t}$.

3. Цикл по i от n до $(t+1)$:
   3.1. Если $x_i = y_t$, $q_{i-t-1} \leftarrow b-1$, иначе $q_{i-t-1} \leftarrow \lfloor (x_i b + x_{i-1}) / y_t \rfloor$.
   3.2. Пока $(q_{i-t-1}(y_t b + y_{t-1}) > x_i b^2 + x_{i-1} b + x_{i-2})$, $q_{i-t-1} \leftarrow q_{i-t-1} - 1$.
   3.3. $x \leftarrow x - q_{i-t-1} y b^{i-t-1}$.
   3.4. Если $x < 0$, $x \leftarrow x + y b^{i-t-1}$ и $q_{i-t-1} \leftarrow q_{i-t-1} - 1$.
4. $r \leftarrow x$.
5. Вернуть q и r.

Модульное умножение

Модульное умножение – наиболее часто встречающаяся операция в алгоритмах криптографии с открытым ключом. Если выполнять модульное умножение непосредственно, сначала перемножая целые числа, а потом вычисляя остаток от деления, то сложность алгоритма будет определяться процедурой нахождения остатка. Рассмотрим методы модульного умножения, не требующие деления.

Наиболее просто выполняется умножение по модулю чисел p вида $p = 2^n \pm 1$. Для этого нужно вычислить целочисленное произведение сомножителей и представить его в виде $2^n A + B$, где B – младшие n бит произведения. Произведение по модулю p будет равно $B \pm A \pmod p$.

Обобщением этого метода является умножение по модулю чисел $p = 2^n \pm c$ для малого c. В этом случае, если $ab = 2^n A + B$, где $B < 2^n$, то

$$ab \equiv B \pm cA \pmod p.$$

При необходимости применяется рекурсия.

Метод Монтгомери

Этот метод, предложенный П. Монтгомери в 1985 году, не требует выполнения операции деления [3].

Определение 1. Пусть N – нечетное число, по модулю которого выполняется операция умножения. Выберем число R, равное степени числа 2 и превышающее N. Определим N– вычет числа a по модулю N как $aR \pmod N$ и произведение Монтгомери чисел a и b по модулю N как

$$MP(a, b, N, R) \equiv abR^{-1} \pmod N.$$

Для вычисления вычета $TR^{-1} \pmod N$ по заданному числу T воспользуемся преобразованием Монтгомери, которое выполняется следующим алгоритмом.

Алгоритм 1. Вычисление преобразования Монтгомери.

Вход. $T=(t_{2n-1}, t_{2n-2}, \ldots, t_0)_B$, $B = 2^m$ (процессор оперирует со словами длины m битов),

$$R = B^n,\ N = (N_{n-1}, N_{n-2}, \ldots, N_0)_B,$$

$$N<R,\ N' \equiv -N^{-1} \pmod{B},$$

где N' – одиночное слово.

Выход. $TR^{-1} \pmod{N}$.

1. Положить $U \leftarrow T$, $U = (U_{n-1}, U_{n-2}, \ldots, U_0)_B$.
2. Для $i = 0,1, \ldots, n-1$ вычислить $v_i \leftarrow u_i N' \pmod{B}$, $U \leftarrow U + v, NB'$.
3. $U \leftarrow U/R$.
4. Если $U \geq N$, то $U \leftarrow U - N$.
5. Результат: U.

Для вычисления произведения Монтгомери $MP(a, b, N\ R)$ достаточно перемножить операнды a и b как целые числа, а затем выполнить преобразование Монтгомери. Но можно действовать исключительно в терминах N-вычетов. Для вычисления произведения $ab \pmod{N}$ сначала переводят числа a и b в N-вычеты, вычисляя

$$MP(a, R^2 \pmod{N}, N, R) \equiv aR \pmod{N},$$

$$bR = MP(b, R^2 \pmod{N}, N, R) \equiv bR \pmod{N}.$$

Затем находят произведение N-вычетов

$$MP(aR, bR, N, R) \equiv (aR)(bR)R^{-1} \equiv (ab)R \pmod{N}.$$

И наконец, выполняют обратное преобразование N-вычета в число

$$MP(aR, 1, N, R) \equiv (aR)(bR)R^{-1} \equiv (ab)R \pmod{N}.$$

Алгоритм 2. Умножение N-вычетов.

Вход. N-вычеты

$$aR \pmod{N} = (a_{n-1}, a_{n-2}, \ldots a_0)_B,$$

$$bR \pmod{N} = (b_{n-1}, b_{n-2}, \ldots b_0)_B,$$

где $B = 2^m$, $R = B^n$, $N' \equiv -N^{-1} \pmod{B}$.

Выход. N-вычет c = $(ab)R \pmod{N}$.

1. Положить $c \leftarrow 0$, $c = (c_{n-1}, c_{n-2}, \ldots c_0)_B$.

2. Для $i = 0, 1, \ldots, n-1$ выполнить следующие действия.

2.1. Положить $u_i \leftarrow (c_0 + a_i b_0)\, N` \pmod{B}$.

2.2. Вычислить $c = \dfrac{(c + a_i (bR \pmod{N})) + u_i N}{B}$.

3. При $c \geq N$ положить $c \leftarrow c - N$.

4. Результат: c.

Заметим, что сумма N-вычетов также является N-вычетом

$$AR + BR = (A + B)R \pmod{N},$$

поэтому множество N-вычетов с операциями умножения по Монтгомери и сложения является кольцом.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Айерлэнд, К. Классическое введение в современную теорию чисел / К. Айерлэнд, М. Роузен. – М. : Мир, 1987. – 315 с.

2. Виноградов, И. М. Основы теории чисел / И. М. Виноградов. – М. : Наука, 1981. – с. 167.

3. Маховенко, Е. Б. Теоретико-числовые методы в криптографии / Е. Б. Маховенко. – М. : Гелиос АРВ, 2006. – 320 с.

4. Смарт, Н. Криптография / Н. Смарт. – М. : Техносфера, 2005. – 528 с.

5. Ван Тилборг, Х. К. А. Основы криптологии. Профессиональное руководство и интерактивный учебник / Х. К. А. Ван Тилборг. – М. : Мир, 2006. – 471 с.

6. Алферов, А. П. Основы Криптографии / А. П. Алферов, А. Ю. Зубков, А. С. Кузьмин, А. В. Черемушкин. – М. : Гелиос АРВ, 2001. – 480 с.

7. Черемушкин, А. В. Лекции по арифметическим алгоритмам в криптографии / А. В. Черемушкин. – М. : МЦМНО, 2002. – 104 с.

8. Саломаа, А. Криптография с открытым ключом / А. Саломаа. – М. : Мир, 1996. – 318 с.

9. Нечаев, В. И. Элементы криптографии. Основы теории защиты информации / В. И. Нечаев. – М. : Высшая школа, 1999. – 109 с.

10. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – М. : Триумф, 2003. – 816 с.

11. Василенко, О. Н. Теоретико-числовые алгоритмы в криптографии / О. Н. Василенко. – М. : МЦНМО, 2003. – 328 с.

12. Харин, Ю. С. Математические и компьютерные основы криптологии / Ю. С. Харин, В. И. Берник, Г. В. Матвеев, С. В. Агиевич. – Минск : ООО Новое знание, 2003. – 382 с.

13. Вельшенбах, М. Криптография на Си и С++ в действии : учеб. пособие /М. Вельшенбах. – М. : Триумф, 2004. – 464 с.

14. Коблиц, Н. Курс теории чисел и криптографии / Н. Коблиц. – М. : Научное издательство ТВП, 2001. – 254 с.

15. Некоторые алгоритмы теории чисел / сост. Ю. А. Шитов и др. – Красноярск, ИПЦ КГТУ, 2003. – 43с.

Приложение 1. ГРУППЫ, КОЛЬЦА, ПОЛЯ

Для криптографии алгебра является одним из основных инструментов в теоретических исследованиях и практических построениях криптографических преобразований. Поэтому в прил. 1 даны некоторые основные стандартные понятия и определения, которые используются в алгебре.

Группы

Группой называется множество G, на котором определена ассоциативная бинарная операция $\circ$, содержащее элемент e такой, что для любого элемента $a \in G$ выполняется

$$e \circ a = a \circ e = a,$$

и существует элемент a^{-1} такой, что

$$a \circ a^{-1} = a^{-1} \circ a = e.$$

Указанный элемент e называется **единицей** группы. Элемент a^{-1} называется **обратным** к элементу a. Легко показать, что единица группы единственна и что элемент, обратный к данному элементу, также определяется однозначно. Если операция $\circ$ коммутативна, то группа называется **коммутативной**, или **абелевой.**

Заметим, что не все коммутативные операции ассоциативны. Например, если в качестве операции $\circ$ использовать среднее арифметическое двух вещественных чисел, то эта коммутативная операция не ассоциативна. Действительно:

$$(a \circ a) \circ b = a \circ b = (a + b)/2,$$

$$a \circ (a \circ b) = a \circ ((a + b)/2) = (3a + b)/4.$$

Примерами абелевых групп являются:

- множества Z целых, Q рациональных, R действительных и C комплексных чисел с соответствующими операциями сложения;
- множества $Q\backslash\{0\}$, $R\backslash\{0\}$ и $C\backslash\{0\}$ отличных от нуля рациональных, действительных и комплексных чисел с соответствующими операциями умножения.

В теории групп используется две равноправных и эквивалентных друг другу терминологических системы: аддитивная и мультипликативная. В аддитивной системе групповую операцию называют операцией сложе-

ния, а группы в аддитивной записи для краткости иногда будем назвать **аддитивными**. Группы, операцию в которых называют умножением, иногда именуются **мультипликативными**.

Операцию аддитивной группы принято обозначать знаком +, операцию мультипликативной группы обозначают знаком умножения x, или • **(**или ее обозначение, по умолчанию, опускают).

Единичный элемент аддитивной группы обозначается 0 и называется **нулем**. Элемент аддитивной группы, обратный к элементу a, обозначается $-a$ и называется **противоположным** к этому элементу. Единичный элемент мультипликативной группы обозначается 1 и называется **единицей**. А элемент о**братный** к a обозначается через a^{-1}.

Ассоциативность операции $\circ$ позволяет записывать кратное произведение

$$(\cdots((a \circ a) \circ\ a) \circ\cdots a) \circ a,$$

опуская скобки

$$a \circ a \circ a \circ \ldots \circ a.$$

Такая формула называется ***k*-ой степенью** элемента a. В аддитивных группах k-я степень элемента a обозначается $k \cdot a$, в мультипликативных группах используется обозначение a^k. По определению,

$$0 \cdot a = 0 \text{ и } a^{(0)} = 1.$$

В аддитивной группе определяется операция **вычитания** $a - b$ по следующей формуле

$$a - b = a + (-b).$$

Результат $a - b$ называется **разностью** элементов a и b.

В мультипликативной группе определяется операция **деления** a/b по следующей формуле:

$$a/b = a \cdot b^{-1}.$$

Результат a/b, или называется **частным** от деления элемента a на элемент b.

Рассмотренные выше примеры аддитивных и мультипликативных групп представляют бесконечные группы. Группа, определенная на конечном множестве G, называется **конечной**. **Тривиальная (единичная)** группа определена на одноэлементном множестве $\{e\}$ и содержит только единицу. Число элементов конечныой группы называется **порядком** груп-

пы. Порядок тривиальной группы равен 1, простейшая нетривиальная группа имеет порядок 2.

Порядком элемента g группы G называется наименьшее число n такое, что $g^n = e$. Порядок элемента g иногда далее обозначается ord g. Элемент, порядок которого равен порядку группы, если он существует, называется **образующим** (**примитивным**) элементом группы. Все ненулевые элементы группы могут быть представлены в виде степени образующего (примитивного) элемента. Группа, имеющая образующий элемент, называется **циклической**.

Отношением эквивалентности называется бинарное отношение, которое обладает свойствами транзитивности, рефлексивности и симметричности. Множество, на котором задано это отношение, разбивается на классы эквивалентности $[a]$, где a – представитель класса. Совокупность классов эквивалентности есть **фактор-множество** множества, на котором определено это бинарное отношение эквивалентности.

Так, на множестве Z целых чисел относительно натурального числа m можно определить отношение

$$\{(x,y) \setminus x \equiv y \bmod m\},$$

где $x \equiv$ y mod m означает, что число m делит разность $x - y$. Это отношение называется **отношением сравнения по модулю *m***, а классы эквивалентности по этому отношению – **классами, или классами вычетов по модулю *m***.

Фактор-множество по этому отношению сокращенно обозначают Z_m, аналогично, классы конгруэнтности обозначаются просто $[a]_m$.

Легко видеть, что

$$x \equiv y \bmod m$$

тогда и только тогда, когда

$$x \bmod m \equiv y \bmod m,$$

где $x \bmod m$ и $y \bmod m$ – остаток от деления числа x или числа y на m.

На фактор-множестве Z_m можно определить арифметические операции. Сумму классов эквивалентности определяют следующим образом:

$$[x]_m + [y]_m = [x + y]_m.$$

Удобно в качестве представителей классов $[x]_m$ использовать наименьшие неотрицательные элементы $x \bmod m$ классов. Тогда операцию сложения можно описать в обозначениях этих представителей:

$$[x]_m + [y]_m = [(x + y) \bmod m]_m.$$

Легко показать, что фактор множество Zm с только что описанной операцией сложения есть аддитивная группа с нулевым элементом $[0]_m$ и что противоположный к элементу $[a]_m$ группы есть элемент $-[a]_m = [m-a]_m$. Аналогично, на фактор-множестве Z_m вводится операция умножения по модулю m, т. е.

$$[x]_m \cdot [y]_m = [x \cdot y]_m = [(x \cdot y) \bmod m]_m.$$

При этом множество ненулевых классов конгруэнтности $[a]_m$, $a \neq 0$, имеющих обратный класс $[a^{-1}]_m$, где

$$a \cdot a^{-1} \bmod m \equiv 1,$$

образует мультипликативную группу, которая обозначается Z_m^*.

Мультипликативной единицей является класс $[1]_m$. Множество классов, взаимно простых с модулем m, как раз и представляют группу Z_m^*. Порядок группы Z_m^* обозначается $\varphi(m)$, так определенная функция называется функцией Эйлера.

Группы Z_m^* и Z_m совпадают тогда и только тогда, когда m – простое число. Рассмотренные аддитивная и мультипликативная группы Z_m и Z_m^* изоморфны аддитивной и мультипликативной группам, заданным на множестве наименьших неотрицательных представителей этих классов, в соответствии с операциями сложения и умножения по модулю m на множестве этих представителей. Поэтому часто вместо группы на фактор-множестве рассматривают группы на множестве представителей классов, при этом представителей этих множества $Z_m = \{0, 1, ..., m-1\}$ и $Z_m^* = \{[a], a$ и m взаимно просты$\}$ обозначают так же, как множества классов.

Подмножество H группы G, замкнутое относительно операций группы и являющееся группой с этими же операциями, называется **подгруппой** данной группы. Например, тривиальная группа есть подгруппа любой группы.

Теорема (Лагранжа). Если m – порядок группы G, а n – порядок элемента $g \in G$, то

$$g^m = e$$

и $m \equiv 0 \bmod n$ (порядок элемента делит порядок группы).

Теорема. Для любой группы G при любом натуральном k порядок t элемента a^k определяется равенством

$$t = \delta/(k, \delta),$$

где $\delta = \operatorname{ord} a$. В частности, $\operatorname{ord} a^k = \delta$ тогда и только тогда, когда $(k,\delta) = 1$ (k и δ – взаимно просты).

Кольца. Поля. Многочлены над полем

Кольцом называется множество R с операциями сложения и умножения такими, что R является абелевой группой относительно сложения и умножения ассоциативна и дистрибутивна относительно операции сложения:

$$(a \cdot b) \cdot c = a \cdot (b \cdot c),$$

$$a \cdot (b + c) = a \cdot b + a \cdot c,$$

$$(b + c) \cdot a = b \cdot a + c \cdot a.$$

Следствием определения кольца является свойство – для любого a

$$a \cdot 0 = 0 \cdot a = 0.$$

Примерами колец служат множества Z целых, Q рациональных и R действительных чисел с операциями сложения и умножения. Кольцо, в котором из уравнения $a \; Ч \; b = 0$ следует, что $a = 0$ или $b = 0$, называется областью **целостности**. Если в кольце имеется мультипликативная единица, то кольцо называется **кольцом с единицей**. Ниже рассматриваются только кольца с единицей.

Элемент a' кольца с единицей такой, что $a \cdot a' = 1$, называется **обратным к элементу** a. Элемент, обратный к элементу a кольца, обозначается a^{-1}. Каждый элемент кольца имеет не более одного обратного к нему элемента. Элемента, обратного к нулевому элементу кольца, не существует. Множество элементов кольца, имеющих обратный элемент, составляет мультипликативную группу кольца R, которая обозначается R^*.

Полем называется кольцо F с единицей, множество ненулевых элементов которого с операцией умножения является абелевой группой. Эта группа называется мультипликативной группой поля. Примерами бесконечных полей являются поля Q рациональных, R действительных и C комплексных чисел.

Подмножество F поля Q, замкнутое относительно обеих операций и являющееся полем, называется **подполем**, что обозначается $F \subseteq Q$. Поле, которое не имеет подполя, не совпадающего с самим полем, называется **простым** полем. Существует единственное простое бесконечное поле – это поле Q рациональных чисел.

Конечные поля называются **полями Галуа** – по имени французского математика Эвариста Галуа. Далее рассматриваются и используются, как правило, конечные поля.

Порядком поля называется число элементов. Конечное поле порядка q обозначается $GF(q)$, или F_q.

Пример. Простейшим полем является поле из двух элементов – поле $GF(2)$. Операции этого поля определяются таблицами, из которых следует, что сложение соответствует булевой функции сложения по модулю 2, а умножение – конъюнкции.

+	A		·	A	
B	0	1	b	0	1
0	0	1	0	0	0
1	1	0	1	0	1

Пример. Если p – простое число, то целые числа $\{0, 1, 2, \ldots, p-1\}$ образуют поле $G(p)$. При этом все операции (сложения, вычитания, умножения, деления) выполняются по модулю p.

Пример. Если p – простое число, а n – натуральное, то поле, которое содержит $N = p^n$ элементов, не может быть образовано из совокупности целых чисел по модулю N. Действительно, для $p = 2$ и $n = 2$ число элементов $N = p^n = = 2^2 = 4$. В множестве классов вычетов по модулю 4 элемент 2 не имеет обратного, так как $2 \cdot 2 = 0 \bmod 4$. То есть множество, состоящее из четырех элементов, совсем не похоже на поле $G(N)$, которое состоит из $p^n = N$ элементов. Элементами поля, которое состоит из p^n элементов, являются все многочлены степени не более $(n-1)$ с коэффициентами из поля $G(p)$. Чтобы подчеркнуть эту разницу между представленными полями, поля из p^n элементов обозначают через $G(p^n)$ (вместо $G(N)$). Поля $G(p^n)$ будут рассматриваться ниже. Мультипликативная группа конечного поля порядка q обозначается $GF(q)^*$ и имеет порядок на единицу меньше порядка поля.

Два поля – $F^{(1)}$ и $F^{(2)}$ – называются **изоморфными**, если существует биекция φ: $F^{(1)} \leftarrow F^{(2)}$, сохраняющая операции. Эта биекция и обратная к ней функция φ^{-1} называются **изоморфизмами**.

Заметим, что фактор-множество Z_p кольца Z целых чисел по модулю простого числа p является полем порядка p и что все конечные поля простого порядка p являются простыми и изоморфны друг другу, т. е. такие поля составляют класс всех простых конечных полей.

Определение. Пусть F – подполе поля H и α – некоторый элемент поля H ($\alpha \in H$). Минимальное поле, которое содержит поле F и элемент α, называют **простым расширением поля** ***F***, которое получено присоединением к полю F элемента α. Это расширение будем обозначать через $F(\alpha)$.

Понятие поля позволяет вводить и использовать большое разнообразие колец, элементы которых определяются как многочлены

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

с коэффициентами a_i из данного поля F. Такие многочлены называются **многочленами над полем** F. Наибольшее число d такое, что коэффициент

$a_d \neq 0$ называется **степенью** многочлена $f(x)$. Если при этом $a_d = 1$, то многочлен степени d называется **нормированным**. Степень многочлена $f(x)$ обозначается $\deg f(x)$. Число ненулевых коэффициентов многочлена будем называть его **весом**. Степень нулевого многочлена естественно определить как −1. Многочлен степени не более $n - 1$ над полем F представим упорядоченным набором $(a_0, a_{1\ldots}, a_{n-1})$ коэффициентов. Иногда удобно использовать наборы длины, полученные добавлением старших нулевых элементов a_i, $i > \deg(x)$.

Кольцо многочленов над полем F образуется всеми многочленами над F. Оно обозначается $F[x]$. Операции сложения и умножения кольца $F[x]$ определяются теми же правилами, по которым складываются или перемножаются многочлены над действительным полем. Операция **сложения** сопоставляет двум многочленам $p_1(x)$ и $p_2(x)$ вида

$$p_1(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1} + a_nx^n$$

и

$$p_2(x) = b_0 + b_1x + \ldots + b_{n-1}x^{n-1} + b_nx^n$$

их сумму

$$p_1(x) + p_2(x) = (a_0 + b_0) + (a_1 + b_1)x + \ldots + (a_{n-1} + b_{n-1})x^{n-1} + (a_n + b_n)x^n.$$

Здесь и ниже в слагаемых формул, подобных формуле в правой части, имеются в виду операции сложения и умножения в поле F. Результатом операции **умножения** двух многочленов $p_1(x)$ и $p_2(x)$ является многочлен

$$p(x) = p_1(x) \cdot p_2(x) = c_0 + c_1x + \ldots + c_{2m-3}x^{2m-3} + c_{2m-2}x^{2m-2},$$

где $c_i = a_tb_l$, $t + l = i$.

Нулем кольца многочленов является многочлен 0, все коэффициенты которого нулевые, т. е. равны аддитивной единице поля. Единицей кольца многочленов является многочлен 1 нулевой степени, Кольцо многочленов не является полем, так как не всякий многочлен имеет обратный к нему элемент кольца. Кольцо многочленов над полем F будем обозначать через $F[x]$.

Говорят, что многочлен g делит многочлен f, если существует многочлен h такой, что $f = g \cdot h$, где $f, g, h \in F[x]$.

Наибольшим общим делителем (НОД) двух многочленов f и g кольца $F[x]$ называется нормированный многочлен наибольшей степени, который делит каждый из этих многочленов.

Многочлен $f \in F[x]$ называется **неприводимым**, если $f = g \cdot h$ только в том случае, когда либо g либо h является константой. Неприводимые

многочлены среди многочленов играют ту же роль, что и простые числа среди целых чисел. Каждый многочлен из кольца может быть представлен в виде произведения неприводимых многочленов единственным образом. Знание неприводимых многочленов над конечными полями нужно, по крайней мере, для того, чтобы строить эти поля. Если p – простое число, $G(p)$ – конечное поле с операциями по модулю p, n – натуральное число, $g(x)$ – неприводимый многочлен степени n, то конечное поле $G(p^n)$, которое состоит из $N = p^n$ элементов, состоит из многочленов степени $n - 1$. В этом случае результат **произведения элементов** $f_1(x)$ и $f_2(x)$ поля $G(p^n)$ – это **остаток от деления** многочлена $f(x) = f_1(x) \cdot f_2(x)$ на неприводимый многочлен $g(x)$. То есть

$$f_1(x) \cdot f_2(x) \equiv r(x) \bmod g(x),$$

где $r(x)$ – остаток от деления $f_1(x) \cdot f_2(x)$ на $g(x)$.

Определение 2. Пусть F – подполе поля H и $f(x)$ – многочлен над полем F. Обозначим через α корень многочлена $f(x)$ в поле H (заметим, что не в F). Тогда простое расширение $F(\alpha)$ (см. определение 1) называют **простым алгебраическим расширением**, которое получается путём присоединения к полю F корня α многочлена f. Если многочлен f – неприводимый над полем F многочлен степени n, то расширение $F(\alpha)$ называют **простым алгебраическим расширением поля *F* степени *n*.**

В последнем случае любой элемент $u \in F(\alpha)$ можно представить как линейную комбинацию степеней α^i, $i = 0, 1, \ldots, n - 1$, т. е. u представимо в виде

$$w = c_0 + c_0 a + \ldots + c_{n-1} a^{n-1},$$

где c_i, $i = 0, 1, \ldots, n - 1$ – элементы поля F.

Полем разложения многочлена f над полем F называют минимальное расширение H поля F, в котором многочлен f разлагается в произведение линейных над полем H многочленов. Справедливо следующее утверждение.

Теорема. Пусть f – многочлен над полем F. Тогда

- существует поле разложения многочлена f над полем F;
- любые поля разложения многочлена f над полем F изоморфны.

Пример. Если Q – поле, состоящее из всех рациональных чисел, тогда $Q(\sqrt{2})$ есть поле разложения многочлена $f(x) = x^2 - 2$. Чтобы получить поле разложения многочлена $f(x) = x^3 - 2 = (x -)(x^2 + x\sqrt{2} + 1)$, надо к полю Q присоединить корни уравнения $x^3 - 2 = (x -)(x^2 + x\sqrt{2} + 1) = 0$.

Справедливо следующее утверждение для полей $G(p^n)$ (p – простое число, p^n – число элементов в поле).

Теорема. Если $G(p^n)$ есть поле с $N = pn$ элементами, то каждый элемент его удовлетворяет уравнению $x^N - x = 0$ и $G(p^n)$ – это в точности множество корней этого уравнения. Обратно, для любой степени простого числа p ($N = p^n$) поле разложения над $F(p)$ многочлена $x^N - x = 0$ есть поле из N элементов.

Среди неприводимых многочленов особую роль играют **примитивные многочлены**, корни которых являются примитивными (порождающими) элементами поля разложения этого многочлена. Примитивные многочлены широко используются в криптографии для построения линейных рекуррентных последовательностей (ЛРП) максимально возможной длины. ЛРП применяются, например, при построении датчиков случайных чисел.

Во многих книгах даны таблицы неприводимых многочленов невысоких степеней. Более того, ввиду важности для приложений неприводимые и примитивные многочлены уже давно табулированы и сведены в таблицы, имеющиеся в некоторых учебниках.

Напомним, что элемент $\alpha \in F$ называется корнем многочлена $f \in F(x)$, если $f(\alpha) = 0$. Имеется связь между делимостью многочленов и корнями. Эта связь формулируется теоремой Безу.

Теорема (теорема Безу). Элемент α поля F тогда и только тогда будет корнем многочлена $f \in F(x)$, когда $x - \alpha$ делит $f(x)$.

Приложение 2. РЕАЛИЗАЦИЯ АЛГОРИТМОВ

Описание

Каждая из приведенных ниже программ должна содержать функцию NumberIn() – функцию ввода числа с обработкой ошибок. Но чтобы не печатать ее в каждой программе (с целью экономии места, да и вообще это будет избыточно), ее нужно включить лишь в ту программу, которая будет первой по порядку в учебном пособии, и указать, что во всех остальных программах она идентична.

```
//Функция ввода числа с обработкой ошибок
long NumberIn ()
{
	long a = 0, s = 0, i = 0;
	char c;
	while (1) {
		c = getch();
		if (c == 13)  //Если нажат клавиша «Enter»
			break;
		if ( !i && !s && c == '-') //Обработка отрицательных чисел
			s = 1, cout << c;
		if ((a == 214748364 && atoi(&c) >= 8) || (a > 214748364))
			break;		//Проверка попадания числа в тип long
		if (isdigit (c)) {	//Формирование числа
			a = a * 10;
			a = a + atoi(&c);
			cout << atoi(&c);
			i = 1;
		}
	}
	if (s)		//Если число отрицательное
		a = - a;
	return a;
}
```

Решение сравнения

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <math.h>
```

```
long GCD (long a, long b);
long Extended_GCD (long a, long b, long &x, long &y);
long Euler_Function (long n);
long Congruence_Solve (long a, long b, long m);
long Congruence_Solve2 (long a, long b, long m);
long NumberIn ();

int main()
{
while (1) {
	clrscr();
	long a, b, m, x;
	cout << "\tПрограмма решения сравнения вида ax = b mod m\n";
	cout << "\n\tВведите число a: ";
	a = NumberIn ();
	cout << "\n\tВведите число b: ";
	b = NumberIn ();
	cout << "\n\tВведите число m: ";
	m = NumberIn ();
	cout << "\n\n\Вы ввели:";
	cout << "\n\ta = " << a;
	cout << "\n\tb = " << b;
	cout << "\n\tm = " << m;
	if (m <= 0)
		cout << "\n\n\tЧисло m должно быть положительным!\n";
	else {
		x = Congruence_Solve (a, b, m);
		switch (x) {
		case -1:
			cout << "\n\n\tСравнение " << a << "x = " << b;
			cout << "mod" << m << " решений не имеет.\n";
			break;
		case -2:
			cout << "\n\n\tСравнение " << a << "x = " << b;
			cout << "mod" << m << " имеет решение x = любое\n";
			break;
		default:
			cout << "\n\n\tСравнение " << a << "x = " << b;
			cout << "mod" << m << " имеет решение x = ";
			cout << x << "\n";
			break;
		}
	}
```

```
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}


//Функция вычисления НОД
long GCD (long a, long b)
{
        return b ? GCD (b, a % b) : a;
}


//Расширенный алгоритм Евклида
long Extended_GCD (long a, long b, long &x, long &y)
{
        if (a < b)                              //Для работы алгоритма a должно
                Extended_GCD (b,a,y,x);         //быть больше b
        else {
                if (b == 0) {           //Если b = 0, НОД(a,b) = a
                        x = 1;          //y может быть любым числом
                        y = 0;
                        return a;
                }
                else {
                        long x1, x2, y1, y2, q, r, xx, yy;
                        x1 = 0; x2 = 1;                 //Начальная инициализация
                        y1 = 1; y2 = 0;
                        while (b > 0) {
                                q = a / b;      //Следующие четыре строки -
                                r = a % b;      //работа «нерасширенного»
                                a = b;          //алгоритма Евклида
                                b = r;          //

                                xx = x2 - q * x1;   //работа расширенного
                                yy = y2 - q * y1;   //алгоритма -
                                x2 = x1;            //Пересчет значений
                                x1 = xx;            //для получения x и y
                                y2 = y1;
                                y1 = yy;
```

```
                }
                x = x2;
                y = y2;
                return a;       //НОД(a,b) = a
            }
        }
    }



    //Функция Эйлера
    long Euler_Function (long n)
    {
        long i, phi = 0;
        for (i = 1; i < n; i++)       //цикл от 1 до n
            if (GCD (i, n) == 1)        //если числа i и n взаимно просты,
                phi++;                  //то инкрементировать phi
        return phi;
    }



    //Решение сравнения с помощью функции Эйлера
    long Congruence_Solve (long a, long b, long m)
    {
        long i, c = 1, d;
        d = GCD (a, m);             //Вычисляем НОД(a,m)
        if (b % d != 0)             //Если d не делит b,
            return -1;      //то сравнение решений не имеет
        if ((a == 0 && b % m == 0) || (b == 0 && a % m == 0))
            return -2;      //Обработка случаев, когда x может быть любым
        if (d != 1) {           //Если НОД(a,m) нетривиален и d делит b, то
            a = a / d;      //делим все коэффициенты сравнения на d
            b = b / d;
            m = m / d;
        }
        for (i = 0; i < Euler_Function (m) - 1; i++)    //Вычисляем функ-
цию Эйлера
            c = (c * a) % m;  //и a^(φ(m)-1) mod m и заносим в переменную c
        c = (c * b) % m;    //вычисляем a^(-1)b mod m и заносим в c
        return c;               //возвращаем решение сравнения
    }
```

```
//Решение сравнения с помощью расширенного алгоритма Евклида
long Congruence_Solve2 (long a, long b, long m)
{
        long d, x, y;
        d = Extended_GCD (a, m, x, y); //вычисляем НОД(a,m), а также
числа x //и y такие, что ax + my = НОД(a,m)
        if (b % d != 0)       //Если d не делит b,
                return -1;    //то сравнение решений не имеет
        if ((a == 0 && b % m == 0) || (b == 0 && a % m == 0))
                return -2;    //Обработка случаев, когда x может быть любым
        if (d != 1) {         //Если НОД(a,m) нетривиален и d делит b, то
                a = a / d;    //делим все коэффициенты сравнения на d
                b = b / d;
                m = m / d;
        }
        d = (b * x) % m;      //вычисляем bx mod m, где x ≡ a^-1 mod m найдено
                              //с помощью расширенного алгоритма Евклида
        return d >= 0 ? d : d + m;//Так как число x, которое получили из
                              //алгоритма Евклида могло быть отрицательным,
                              //проверяем на отрицательность полученное
                              //решение. И если оно меньше нуля, то
                              //увеличиваем его на m.
}
```

Алгоритм Евклида

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>

long GCD (long a, long b);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long a, b, gcd;
        cout << "\tПрограмма вычисления НОД(a,b) по алгоритму Евк-
лида.\n";
        cout << "\n\tВведите число a: ";
        a = NumberIn ();
        cout << "\n\tВведите число b: ";
```

```
        b = NumberIn ();
        cout << "\n\tВы ввели:";
        cout << "\n\ta = " << a;
        cout << "\n\tb = " << b;
        if (a < 0 || b < 0)
                cout << "\n\n\tЧисла a и b должны быть положительными!\n";
        else
                if ((gcd = GCD (a, b)) != 0)
                        cout << "\n\n\tНОД(" << a << ", " << b << ") = ";
                        cout << gcd << "\n";
                else
                        cout << "\n\n\tНОД(0, 0) не определен!\n";
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}


//Функция вычисления НОД
long GCD (long a, long b)
{
        if (a < b)              //Если a < b, то
                GCD (b,a);  //вызываем функцию НОД (b,a)
        else {
                if (b == 0)             //Если b = 0, то НОД = a.
                        return a;
                long q, r = 1;          //r = 1 чтобы цикл выполнился хотя бы
1 раз
                while (r != 0) {        //пока получаем ненулевые остатки
                        q = a / b;
                        r = a % b;
                        a = b;
                        b = r;
                }
                return a;               //последний ненулевой остаток будет
                                        //находиться в переменной a
        }
}
```

Бинарный алгоритм Евклида

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>

long GCD_binary (long a, long b);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long a, b, gcd;
        cout << "\tПрограмма вычисления НОД(a,b) по алгоритму Евклида.\n";
        cout << "\n\tВведите число a: ";
        a = NumberIn ();
        cout << "\n\tВведите число b: ";
        b = NumberIn ();
        cout << "\n\tВы ввели:";
        cout << "\n\ta = " << a;
        cout << "\n\tb = " << b;
        if (a < 0 || b < 0)
                cout << "\n\n\tЧисла a и b должны быть положительными!\n";
        else
                if ((gcd = GCD_binary (a, b)) != 0)
                        cout << "\n\n\tНОД (" << a << ", " << b << ") = ";
                        cout << gcd << "\n";
                else
                        cout << "\n\n\tНОД (0, 0) не определен!\n";
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}
```

```
//Вычисление НОД (Бинарный алгоритм Евклида)
long GCD_binary (long a, long b)
{
	if (a < b)	//Для работы алгоритма необходимо, чтобы
		GCD_binary (b,a);// а было больше, чем b
	else {
		if (b == 0)			//Обработка «крайних» случаев
			return a;	// (когда одно из чисел равно нулю)
		long g = 1;
		while (a % 2 == 0 & b % 2 == 0) {	//пока оба числа четные
			a = a / 2;
			b = b / 2;
			g = g * 2;
		}
		long u = a, v = b;
		while (u != 0) {
			while (u % 2 == 0) {		//пока u четно
				u = u / 2;
			}
			while (v % 2 == 0) {		// пока u четно
				v = v / 2;
			}
			if (u >= v)		//присваиваем положительную
				u = u - v;	//разность u и v одной из
			else				// одной из переменных
				v = v - u;
		}
		long d = g * v;			//возвращаем НОД
		return d;
	}
}
```

Расширенный алгоритм Евклида

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>

long NumberIn ();
long Extended_GCD (long a, long b, long &x, long &y);

int main()
```

```
{
while (1) {
	clrscr();
	long a, b, x, y, gcd;
	cout << "\tПрограмма вычисления НОД(a,b) и чисел x и y\n";
	cout << "\tпо расширенному алгоритму Евклида.\n";
	cout << "\n\tВведите число a: ";
	a = NumberIn ();
	cout << "\n\tВведите число b: ";
	b = NumberIn ();
	cout << "\n\n\tВы ввели:";
	cout << "\n\ta = " << a;
	cout << "\n\tb = " << b;
	if (a < 0 || b < 0)
		cout << "\n\n\tЧисла a и b должны быть положительными!\n";
	else
		if ((gcd = Extended_GCD (a, b, x, y)) != 0) {
			cout << "\n\n\tНОД (" << a << ", " << b << ") = ";
			cout << gcd;
			cout << "\n\tx = " << x;
			cout << "\n\ty = " << y << "\n";

			cout << "\n\tПроверка:";
			cout << "\n\ta * x + b * y =";
			cout << "\t" << a << " * " << x << " + ";
			cout << b << " * " << y << " = " << (a * x + b * y);
			cout << " = " << "НОД(a,b)\n";
		}
		else
			cout << "\n\n\tНОД (0, 0) не определен!\n";
	cout << "\n\n\tДля выхода нажмите Esc";
	cout << "\n\tДля продолжения нажмите любую другую клавишу!";
	char c = getch();
	if (c == 27)
		break;
}
return 0;
}
long Extended_GCD (long a, long b, long &x, long &y)
{
	if (a < b)				//Для работы алгоритма a должно
		Extended_GCD (b,a,y,x);		//быть больше b
	else {
```

```
            if (b == 0) {          //Если b = 0, НОД(a,b) = a
                x = 1;             //y может быть любым числом
                y = 0;
                return a;
            }
            else {
                long x1, x2, y1, y2, q, r, xx, yy;
                x1 = 0; x2 = 1;                 //Начальная инициализация
                y1 = 1; y2 = 0;
                while (b > 0) {
                    q = a / b;      //Следующие четыре строки -
                    r = a % b;      //работа «нерасширенного»
                    a = b;          //алгоритма Евклида
                    b = r;          //

                    xx = x2 - q * x1;   //работа расширенного
                    yy = y2 - q * y1;   //алгоритма -
                    x2 = x1;            //Пересчет значений
                    x1 = xx;            //для получения x и y
                    y2 = y1;
                    y1 = yy;
                }
                x = x2;
                y = y2;
                return a;       //НОД(a,b) = a
            }
        }
}
```

Вычисление символа Якоби

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>

long Jacobi_Symbol (long a, long b);
long NumberIn ();

int main()
{
while (1) {
        clrscr();
        long a, p, J;
```

```
        cout << "\tПрограмма вычисления символа Якоби J(a,p).\n";
        cout << "\n\tВведите число a: ";
        a = NumberIn ();
        cout << "\n\tВведите число p: ";
        p = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\ta = " << a;
        cout << "\n\tp = " << p;
        if (p <= 0 || p % 2 == 0)
                cout << "\n\tЧисло p должно быть положительным и не-
четным!\n";
        else
                cout << "\n\n\tJ (" << a << ", " << p << ") = ";
                cout  << Jacobi_Symbol (a, p) << "\n";
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
    }
    return 0;
    }

    long Jacobi_Symbol (long a, long b)
    {
        if (a == 0)
                return 0;
        if (a == 1)             //если a = 1, то символ Якоби = 1 (свойство 4)
                return 1;
        if (a < 0)              //если a < 0, то если
                if ((b & 2) == 0)    // (p-1)/2 – четное, то вычислить
                        return Jacobi_Symbol (-a, b);    //символ Якоби J(-a,b)
                else
                        return - Jacobi_Symbol (-a, b);//иначе – J(-a,b)
        long a1 = a,  e = 0;
        while ((a1 & 1) == 0)           //представляем число в виде
                a1 >>= 1,  ++e;         //a = 2^t a_1
        long s;
        if ((e & 1) == 0 || (b & 7) == 1 || (b & 7) == 7)    //если e - четное
                                                             (p^2 - 1)/8
                s = 1;                  //или число                    - четное,
        else                            // то s = 1,
```

```
            s = -1;                                 //иначе s = -1
        if ((b & 3) == 3 && (a1 & 3) == 3)      //применяем закон взаим-
ности:
            s = -s;                         //если число (p-1)/2 * (q-1)/2
                                            //нечетное, то s = -s
        if (a1 == 1)                    //если разложение закончилось,
            return s;                   //вернуть значение символа Якоби,
        return s * Jacobi_Symbol (b % a1, a1);              //иначе продол-
жить
                                                    //вычисление
    }
```

Вычисление символа Лежандра

```
    #include <conio.h>
    #include <stdio.h>
    #include <math.h>
    #include <iostream.h>

    int Trial_Division (long n);
    long Jacobi_Symbol (long a, long b);
    long Legendre_Symbol (long a, long b);
    long NumberIn ();

    int main(void)
    {
    while (1) {
        clrscr();
        long a, p, L;
        cout << "\tПрограмма вычисления символа Лежандра L(a,p).\n";
        cout << "\n\tВведите число a: ";
        a = NumberIn ();
        cout << "\n\tВведите число p: ";
        p = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\ta = " << a;
        cout << "\n\tp = " << p;
        if ( (L = Legendre_Symbol (a, p)) == -2)
            cout << "\n\tЧисло p должно быть положительным и про-
стым!\n";
        else
            cout << "\n\n\tL (" << a << ", " << p << ") = " << L << "\n";
```

```
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
    }
    return 0;
    }

    long Jacobi_Symbol (long a, long b)
    {
        if (a == 0)
                return 0;
        if (a == 1)             //если a = 1, то символ Якоби = 1 (свойство 4)
                return 1;
        if (a < 0)              //если a < 0, то если
                if ((b & 2) == 0)    // (p-1)/2 – четное, то вычислить
                        return Jacobi_Symbol (-a, b);   //символ Якоби J(-a,b)
                else
                        return - Jacobi_Symbol (-a, b);//иначе – J(-a,b)
        long a1 = a,  e = 0;
        while ((a1 & 1) == 0)               //представляем число в виде
                a1 >>= 1,  ++e;             // a = 2^t a_1
        long s;
        if ((e & 1) == 0 || (b & 7) == 1 || (b & 7) == 7)    //если e - четное
                s = 1;                      //или число (p^2 - 1)/8 - четное,
        else                                // то s = 1,
                s = -1;                             //иначе s = -1
        if ((b & 3) == 3 && (a1 & 3) == 3)    //применяем закон взаим-
ности:
                s = -s;                     //если число (p-1)/2 * (q-1)/2
                                            //нечетное, то s = -s
        if (a1 == 1)                //если разложение закончилось,
                return s;           //вернуть значение символа Якоби,
        return s * Jacobi_Symbol (b % a1, a1);       //иначе продолжить
                                                     //вычисление
    }

    int Trial_Division (long n)
    {
        long i;
        for (i = 2; i <= sqrt(n); i++)              //цикл от 2 до √n
```

```
            if (n % i == 0)                    //если n делится на i,
                return 0;           //вернуть 0
        return 1;           //если делителей не найдено, вернуть 1
    }

    long Legendre_Symbol (long a, long b)
    {
        if (Trial_Division (b))     //если число p - простое
            return Jacobi_Symbol (a, b);     //вычислить символ Якоби
        else
            return -2;     //иначе вернуть -2 (символ не определен)
    }
```

Решето Эратосфена

```
    #include <conio.h>
    #include <stdio.h>
    #include <iostream.h>

    int *Sieve_Of_Eratosthenes (int n);
    long NumberIn ();

    int main(void)
    {
    while (1) {
        clrscr();
        long n;
        cout << "\tПрограмма нахождения всех простых чисел\n";
        cout << "\tв интервале от 0 до n с помощью решета Эратосфе-
на.\n";
        cout << "\n\tВведите число n: ";
        n = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n;
        if (n <= 0)
            cout << "\n\n\tЧисло n должно быть положительным!\n";
        else {
            int *a;
            a = Sieve_Of_Eratosthenes (n);
            FILE *f = fopen("Sieve.txt","w");
            for (int i = 0, k = 0; i < n+1; i++) {
                if (a[i])
```

```
                        fprintf(f,"%5d  ",i), k++;
                    if (k == 10)
                        fprintf(f,"\n"), k = 0;
                }
                fclose(f);
                cout << "\n\n\tТаблица простых чисел в файле Sieve.txt\n";
        }
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}

int *Sieve_Of_Eratosthenes (int n)
{
        int i, j;
        int *prime;
        prime = new int [n+1];              //массив чисел от 1 до n
        for (i = 0; i < n+1; i++)   //инициализируем единицами
                prime[i] = 1;
        prime[0] = prime[1] = 0; //число 0 не рассматриваем,
                                 //число 1 – ни простое, ни составное
        for (i = 2; i < n+1; i++)   //цикл по всем числам < n
                if (prime[i])           //если число не вычеркнуто,
                        for (j=i+i; j < n+1; j+=i)  //то вычеркиваем все
                                prime[j] = 0;         //числа, кратные ему
        return prime;
}
```

Пробное деление

```
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <math.h>

int Trial_Division (long n);
long NumberIn ();

int main(void)
{
```

```
while (1) {
        clrscr();
        long n;
        cout << "\tТест на простоту методом пробного деления.\n";
        cout << "\n\tВведите число n: ";
        n = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n << "\n\n\t";
        if (n <= 0)
                cout << "Число n должно быть положительным!\n";
        else
                if (n == 1)
                cout << "Число 1 не является ни простым, ни составным.\n";
                else
                        if (Trial_Division (n))
                                cout << "Число " << n << " простое.\n";
                        else
                                cout << "Число " << n << " составное.\n";
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}

int Trial_Division (long n)
{
        long i;
        for (i = 2; i <= sqrt(n); i++)              //цикл от 2 до √n
                if (n % i == 0)                     //если n делится на i,
                        return 0;               //вернуть 0
        return 1;               //если делителей не найдено, вернуть 1
}
```

Тест на основе малой теоремы Ферма

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
```

```
int Fermat_Primality_Test (long n, long k);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long n, k;
        cout << "\tТест на простоту, основанный на малой теореме
Ферма.\n";
        cout << "\n\tВведите число n: ";
        n = NumberIn ();
        cout << "\n\tВведите число k: ";
        k = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n;
        cout << "\n\tk = " << k << "\n\n\t";
        if (n <= 0 || k <= 0)
                cout << "Числа n и k должны быть положительными!\n";
        else
                if (n == 1)
                cout << "Число 1 не является ни простым, ни составным.\n";
                else
                        if (Fermat_Primality_Test (n, k))
                        cout << "Число " << n << " вероятно простое.\n";
                        else
                        cout << "Число " << n << " составное.\n";
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}

int Fermat_Primality_Test (long n, long k)
{
        long i, j;
        for (i = 0; i < k; i++) {     //цикл от 0 до количества попыток к
                long a, c = 1;
                a = random(n-1) + 1;       //выбираем случайное a
                for (j = 0; j < n-1; j++)    //возводим a в степень (n-1)
```

```
                c = (c * a) % n;      //по модулю n
            if (c != 1)               //если c не равно единице (т. е.
                return 0;             //не выполняется малая теорема Ферма),
        }                             //то n - составное
        return 1;       //иначе n - вероятно простое
}
```

Тест Соловея – Штрассена

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

long GCD (long a, long b);
long Jacobi_Symbol (long a, long b);
int SolovayStrassen_Primality_Test (long n, long k);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long n, k;
        cout << "\tТест на простоту Соловея – Штрассена.\n";
        cout << "\n\tВведите число n: ";
        n = NumberIn ();
        cout << "\n\tВведите число k: ";
        k = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n;
        cout << "\n\tk = " << k << "\n\n\t";
        if (n <= 0 || k <= 0)
                cout << "Числа n и k должны быть положительными!\n";
        else
                if (n == 1)
                cout << "Число 1 не является ни простым, ни составным.\n";
                else
                        if (SolovayStrassen_Primality_Test (n, k))
                        cout << "Число " << n << " вероятно простое.\n";
                        else
                        cout << "Число " << n << " составное.\n";
```

```
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
    }
    return 0;
    }

    long GCD (long a, long b)
    {
        return b ? GCD (b, a % b) : a;
    }

    long Jacobi_Symbol (long a, long b)
    {
        if (a == 0)
                return 0;
        if (a == 1)             //если a = 1, то символ Якоби = 1 (свойство 4)
                return 1;
        if (a < 0)              //если a < 0, то если
                if ((b & 2) == 0)    // (p-1)/2 – четное, то вычислить
                        return Jacobi_Symbol (-a, b);    //символ Якоби J(-a,b)
                else
                        return - Jacobi_Symbol (-a, b);//иначе – J(-a,b)
        long a1 = a,  e = 0;
        while ((a1 & 1) == 0)               //представляем число в виде
                a1 >>= 1,  ++e;             //a = 2^t a_1
        long s;
        if ((e & 1) == 0 || (b & 7) == 1 || (b & 7) == 7)    //если e - четное
                                                             (p^2 - 1)/8
                s = 1;                      //или число                - четное,
        else                                // то s = 1,
                s = -1;                             //иначе s = -1
        if ((b & 3) == 3 && (a1 & 3) == 3)    //применяем  закон  взаим-
ности:
                s = -s;                     //если число (p-1)/2 * (q-1)/2
                                            //нечетное, то s = -s
        if (a1 == 1)                  //если разложение закончилось,
                return s;             //вернуть значение символа Якоби,
        return s * Jacobi_Symbol (b % a1, a1);       //иначе продолжить
                                                     //вычисление
    }
```

```
int SolovayStrassen_Primality_Test (long n, long k)
{
	long i, j;
	if (n == 2)		//число 2 - простое
		return 1;
	if (n % 2 == 0)			//если число четное,
		return 0;	//то оно составное
	for (i = 0; i < k; i++) {	//цикл по количеству попыток к
		long a = random(n-1) + 1;	//выбираем случайное a
		if (GCD (a, n) != 1)	//если НОД (a, n) не тривиален,
			return 0;	//то число n составное
		long c = 1;
		for (j = 0; j < (n - 1) / 2; j++)	//вычисляем a^((n-1)/2) mod n
			c = (c * a) % n;
		if (c == n - 1)	//поправляем значение c, чтобы можно
			c = -1;	//было сравнивать с символом Якоби
		if (c != Jacobi_Symbol(a, n))	//если c не равно символу
Якоби,
			return 0;	//то число n составное
	}
	return 1;	//число n – вероятно простое
}
```

Тест Рабина – Миллера

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

long GCD (long a, long b);
int MillerRabin_Primality_Test (long n, long k);
long NumberIn ();

int main(void)
{
while (1) {
	clrscr();
	long n, k;
	cout << "\tТест на простоту Рабина – Миллера.\n";
	cout << "\n\tВведите число n: ";
```

```
        n = NumberIn ();
        cout << "\n\tВведите число k: ";
        k = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n;
        cout << "\n\tk = " << k << "\n\n\t";
        if (n <= 0 || k <= 0)
                cout << "Числа n и k должны быть положительными!\n";
        else
                if (n == 1)
                cout << "Число 1 не является ни простым, ни составным.\n";
                else
                        if (MillerRabin_Primality_Test (n, k))
                        cout << "Число " << n << " вероятно простое.\n";
                        else
                                cout << "Число " << n << " составное.\n";
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}

long GCD (long a, long b)
{
        return b ? GCD (b, a % b) : a;
}

int MillerRabin_Primality_Test (long n, long k)
{
        long i, j;
        if (n == 2)             //число 2 - простое
                return 1;
        if (n % 2 == 0)         //четное число - составное
                return 0;
        long s = 0,
            t = n - 1;          //представляем число n - 1
        while (t % 2 == 0) {        //в виде n - 1 = 2^s t
                t = t / 2;
                s++;
        }
```

```
        for (i = 0; i < k; i++) {           //цикл по количеству попыток
                long a = random(n-1) + 1;//выбираем случайное a
                if (GCD(a, n) != 1)         //если НОД (a, n) не тривиален,
                        return 0;           //то n - составное
                long c = 1;
                for (j = 0; j < t; j++)     //вычисляем a^t mod n
                        c = (c * a) % n;
                if (c == 1 || c == n-1)     //если c = ± 1 mod n, то
                        continue;           //выбираем другое a
                int f = 0;              //флаг = 0
                //вычисляем последовательность
                //(a^t)^2 mod n,(a^t)^4 mod n,...,(a^t)^(2^(s-1)) mod n
                for (j = 1; j <= s-1; j++) {
                                c = (c * c) % n;
                                if (c == n - 1)     //если встретилась -1,
                                f = 1;              //то флаг = 1
                }
                if (f == 0)             //если не было -1,
                        return 0;       //то n - составное
        }
        return 1;       //n – вероятно простое
}
```

Метод p–1 Полларда

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

long GCD (long a, long b);
long Pollard_p1_algorithm (long n);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long n;
        cout << "\tФакторизация методом p-1 Полларда.\n";
        cout << "\tПрограмма вычисляет делитель числа n\n";
```

```
            cout << "\t(Необязательно простой!).\n";
            cout << "\n\tВведите число n: ";
            n = NumberIn ();
            cout << "\n\n\tВы ввели:";
            cout << "\n\tn = " << n << "\n\n\t";
            if (n <= 0)
                  cout << "Число n должно быть положительным!\n";
            else {
                  long d = Pollard_p1_algorithm (n);
                  if (d == 1)
                        cout << "Делителей не найдено.\n";
                  else
                        cout << "Найден делитель " << d << ".\n";
            }
            cout << "\n\n\tДля выхода нажмите Esc";
            cout << "\n\tДля продолжения нажмите любую другую клавишу!";
            char c = getch();
            if (c == 27)
                  break;
      }
      return 0;
      }
      long GCD (long a, long b)
      {
            return b ? GCD (b, a % b) : a;
      }
      long Pollard_p1_algorithm (long n)
      {
            long b = 13; //выбираем число b
            long q[] = {2, 3, 5, 7, 11, 13};   //массив простых чисел
            long qn = 6; //размер массива
            long a = 1;   //инициализация числа a – на первой итерации цик-
ла a = 2
            long i, j, k;
            for (j = 0; j < 10; j++) {   //количество попыток
                  a++;   //изменяем число a
                  for (i = 0; i < qn; i++) {   //цикл по всем q[i]
                        //Вычислить e = log B / log q_i
                        long qq = q[i];
                        long e = floor (log((double)b) / log((double)qq));
                        //Вычислить q_i^e
```

```
                    long nn = 1;
                    for (k = 0; k < e; k++)
                            nn = (nn * qq) % n;
                    //Вычислить aa = a^(q_i^e)
                    long aa = 1;
                    for (k = 0; k < nn; k++)
                            aa = (aa * a) % n;
                    if (aa == 0)  //Продолжить поиск с другим q[i]
                            continue;
                    long g = GCD(aa-1, n);   //вычислить НОД
                    if (1 < g && g < n)         //если НОД нетривиален,
                            return g;       //то это элемент разложения
            }
        }
        return 1;       //если ничего не нашли, вернуть 1
}
```

Метод ρ-Полларда

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

long GCD (long a, long b);
long f (long a, long m);
long Pollard_rho_algorithm (long n);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long n;
        cout << "\tФакторизация методом ρ -Полларда.\n";
        cout << "\tПрограмма вычисляет делитель числа n\n";
        cout << "\t(Необязательно простой!).\n";
        cout << "\n\tВведите число n: ";
        n = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n << "\n\n\t";
        if (n <= 0)
                cout << "Число n должно быть положительным!\n";
```

```
        else {
                long d = Pollard_rho_algorithm (n);
                if (d == 1)
                        cout << "Делителей не найдено.\n";
                else
                        cout << "Найден делитель " << d << ".\n";
        }
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клавишу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}

long GCD (long a, long b)
{
        return b ? GCD (b, a % b) : a;
}

long f (long a, long m)
{
        return (a * a + 1) % m;    //функция f(x) = x^2 +1
}

long Pollard_rho_algorithm (long n)
{
        long x, y, gcd, i;
        x = random (n-1) +1;      //выбирается случайное a
        y = x;                    //y = x
        y = f(y,n);               //начальная инициализация
        gcd = GCD (abs(x - y), n);        //вычисляем НОД
        for (i = 0; i < 20 && (gcd == 1 || gcd == n); i++) {
        // пока i < количества попыток и НОД = 1 или n
                x = f(x,n);               //пересчитываем значения
                y = f(f(y,n),n);       // чисел x и y
                gcd = GCD (abs(x - y), n);        //находим НОД
        }
        if (gcd == n)          //если факторизацию провести не удалось,
                gcd = 1;       //возвращаем единицу
        return gcd;            //возвращаем делитель числа n
}
```

Факторизация Ферма (разность квадратов)

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <math.h>

long Fermat_method (long n);
long NumberIn ();

int main(void)
{
while (1) {
        clrscr();
        long n;
        cout << "\tФакторизация Ферма (разность квадратов).\n";
        cout << "\tПрограмма вычисляет делитель числа n\n";
        cout << "\t(Необязательно простой!).\n";
        cout << "\n\tВведите число n: ";
        n = NumberIn ();
        cout << "\n\n\tВы ввели:";
        cout << "\n\tn = " << n << "\n\n\t";
        if (n <= 0)
                cout << "Число n должно быть положительным!\n";
        else {
                long d = Fermat_method (n);
                if (d == 1) {
                        cout << "Делителей не найдено.\n";
                        cout << "\tЧисло " << n << " простое.\n";
                }
                else
                        cout << "найден делитель " << d << ".\n";
        }
        cout << "\n\n\tДля выхода нажмите Esc";
        cout << "\n\tДля продолжения нажмите любую другую клави-
шу!";
        char c = getch();
        if (c == 27)
                break;
}
return 0;
}
```

```
long Fermat_method (long n)
{
    if (n == 2)   //число 2 - простое
        return 1;
    if (n % 2 == 0)          //если число четное, то
        return 2;            //оно составное
    long x, y, diff;
    x = ceil (sqrt ((double)n));      //начальная инициализация пере-
менных
    y = 0;
    diff = x * x - y * y - n;
    for (;;)
        if (diff == 0)       //если diff равна нулю,
            return x!=y ? x-y : x+y;  //вернуть результат
        else
            if (diff > 0)  //если diff положительна,
                diff -= y+y+1, y++;//то инкрементировать y
            else                        //и скорректировать diff
                diff += x+x+1, x++;//иначе инкрементировать x
                                // и скорректировать diff
}
```

ОГЛАВЛЕНИЕ